Angerhausen · Riedner
Schellenberger

VC-20
Tips & Tricks

Eine Fundgrube für den VC-20 Anwender

Ein DATA BECKER BUCH

Angerhausen · Riedner
Schellenberger

VC-20 Tips & Tricks

Eine Fundgrube für den VC-20 Anwender

Ein DATA BECKER BUCH

Copyright (C) 1983 DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

7 Anwenderprogrammierung für Fortgeschrittene

VORWORT

Mit >> VC-20 TIPS & TRICKS << liegt ein neues DATA BECKER BUCH vor Ihnen. Es soll Ihnen helfen, die fantastischen Möglichkeiten von Commodore's Millionending näher zu erschließen. Wie schon bei unseren anderen Büchern, lag unser Schwergewicht deshalb nicht bei bunten Bildern und schriftstellerischer Meisterleistungen. Wir haben uns stattdessen auf handfeste Informationen spezialisiert, die Sie sofort nutzen können. Vor allem gehören dazu zahlreiche, ausführlich dokumentierte Beispielprogramme. Die Autoren dieses Buches sind alle begeisterte Computerfans und -Experten, die hier ihre Erfahrungen wiedergeben. Michael Angerhausen ist Mitglied des DATA BECKER Softwareteams. Sein Schwerpunkt sind kommerzielle Programme und Datenbanksysteme. Axel Riedner arbeitet im Heimcomputer Verkauf bei DATA BECKER. Er kennt aus seiner täglichen Arbeit die Fragen von VC-20 Anwendern und hat versucht, möglich viele davon in diesem Buch zu beantworten. Wolfgang Schellenberger gehört auch zum DATA BECKER Softwareteam. Er beschäftigt sich hauptsächlich mit Sound und Graphik. Unterstützt wurden die drei Autoren von unserem Entwicklungsschef Klaus Gerits und von Lothar Englisch. Nicht unerwähnt bleiben sollten auch die vielen Helfer, ohne die die Erstellung eines solchen Buches kaum noch denkbar erscheint: unsere Computer. VC-20 TIPS & TRICKS wurde komplett mit Textverarbeitung erstellt und auf einem EPSON Drucker ausgedruckt.

Viel Spaß bei der Lektüre dieses Buches.

Kapitel 2 Der Speicher des VC-20 und seine Erweiterung

Wenn Sie einmal den Anzeigenteil der Fachzeitschriften durchgeblättert haben, wird ihnen sicher aufgefallen sein, daß gerade für den VC-20 Speichererweiterungen gigantischer Ausmaße angeboten werden, d.h. 64K und darüber.
In diesem Kapitel wollen wir Ihnen zeigen, in welchem Rahmen derartige Projekte realisierbar sind. Wir wollen Sie in die Lage versetzen, die diversen Angebote richtig zu beurteilen.
Wir beginnen zweckmäßigerweise mit

2.1 Der Speicheraufbau des VC-20

Der VC-20 arbeitet mit einem Prozessor des Typs 6502. Zu dessen Eigenschaften gehört, daß er einen Adressraum von 64K hat, d.h. aufgrund seiner 16 Adressleitungen kann er nur diesen Bereich durchgehend adressieren, und nicht mehr.
Diesen wichtigen Punkt sollten Sie sich merken.
Beim Design eines Rechners mit einem vorgegebenen Prozessor muß man sich natürlich Gedanken machen, wie dieser zur Verfügung stehende Adressraum sinnvoll aufgeteilt wird. Es sind hier nämlich nicht nur Betriebssystem, Basicinterpreter und Ram unterzubringen, sondern auch, aufgrund verschiedener

Eigenarten der Prozessorfamilie 65xx, auch der Ein- Ausgabebereich, also der Teil, der für die Kommunikation mit der Außenwelt (Tastatur, IEC-Bus, Userport usw.) vorgesehen ist.
Im Falle des VC-20 heißt das konkret, daß von dem zunächst genügend groß erscheinenden 64K folgendes abgezogen werden muß:

* 8K Betriebssystem
* 8K Basicinterpreter
* 4K Ein- Ausgabe
* 4K Zeichengenerator

Es bleibt also noch ein Bereich von 40K übrig, der mit Ram bestückt werden könnte.
Zur Darstellung eines Bildes auf dem Schirm wird ein sog. Bildwiederholspeicher benötigt, an anderer Stelle auch Videoram genannt, der eine Größe von 1/2K hat.
Darüberhinaus benötigt das Betriebssystem zum korrekten Arbeiten noch einen Speicher von 1K, in dem die sog. Zeropage, der Stack (Stapelspeicher zur Schachtelung von Unterprogrammen) und weitere Ablaufparameter untergebracht sind.
Außerdem ist noch ein Bereich von 8K reserviert, in dem das Betriebssystem etwa vorhandene Erweiterungen oder Spielmodule erwartet.
Das Videoram, welches sich aufgrund der Hardware des Videocontrollers nur in den unteren 8K befinden darf, begrenzt den für den Anwender zur Verfügung stehenden Speicher nach unten. Nach oben wird der Bereich vom Zeichengenerator abgeschlossen, sodaß letztlich für Ihre

Programme noch ca. 28K zur Verfügung stehen.

Dieser Bereich kann auf verschiedene Weise genutzt werden. In der Grundausstattung enthalten ist ein Ram von 5K. Nach Abzug von Videoram und des vom Betriebssytem benötigten Speichers bleiben ca. 3 1/2K für den Anwender übrig. Hieraus ergibt sich, daß nur Speichererweiterungen von bis zu 24K für reine Basicanwendungen sinnvoll und, was sehr wichtig ist, vom Betriebssystem problemlos zu handhaben sind. Auf diesen Umstand gehen wir im Abschnitt 2.5 noch einmal im Besonderen ein.

Auf den folgenden drei Seiten finden Sie nun die Speicheraufteilungen des VC-20 bei unterschiedlichen Speicherausbauten.
Sollten Sie sich bereits einmal gewundert haben, daß Programme, die in Videoram und Farbram hineinpoken, wohl auf der Grundversion gelaufen sind, jedoch nach einer Erweiterung um 8K ihren Dienst versagen, finden Sie hier die Lösung für diesen Effekt. Sie sollten diesbezüglich einmal die Lage von Videoram und Farbram auf der dritten Abbildung mit der auf den ersten beiden Bildern vergleichen.
An dieser Stelle sei Ihnen verraten, wie Sie Ihre Programme veranlassen können, auf allen Konfigurationen zu laufen:
Beabsichtigen Sie, den Video- oder Farbram mittels POKE zu handhaben, so ermitteln Sie bitte deren Basisadressen folgendermaßen

```
Videoram: BS=4*(PEEK(36866)AND128)+64*(PEEK(36869)AND120)
Farbram : FA=4*(PEEK(36866)AND128)+37888
```

Alle folgenden POKEs, die Farb- oder Videoram zum Ziele haben, sollten sich auf die derart festgestellten Adressen beziehen, etwa in der Form

POKE BS+rel. Position,Bildschirmcode
POKE FA+rel. Position,Farbcode

Die Darstellung der Adressen in den Diagrammen auf den folgenden Seiten erfolgt in der sedezimalen Schreibweise. Wenn Sie damit nicht vertraut sind, brauchen Sie sich nur zu merken, daß ein Sprung um den Wert 1 in der ersten Stelle einem Sprung um 4K im Adressraum entspricht.

Adresse	Bereich
#E000	BETRIEBSSYSTEM
#C000	BASIC
#A000	SPIELMODULE ETC.
#9000	EIN- AUSGABE
#8000	ZEICHENGENERATOR
#2000	FREI
#1E00	VIDEO-RAM
#1000	BASIC-RAM
#0400	FREI
#0000	ZEROPAGE, STACK

2.2 Grundausbau

Diese Speicheraufteilung ist in der Grundausstattung des VC-20 gegeben.
Hierbei fällt auf, daß der Basicbereich nicht sofort oberhalb von Zeropage und Stack beginnt, sondern ein Stück höher.
Das liegt daran, daß der Videoram Bestandteil des auch für Basic verwendeten Speichers ist, und man diesen an einer glatten 8K-Grenze ausgerichtet hat, um spätere externe Erweiterungen nicht durch aufwendige Adressdecodierungen unnötig zu erschweren.
Die mit FREI bezeichnete Lücke von 3K unterhalb des Basicram dient zur Aufnahme einer 3K-Erweiterung.

#E000	BETRIEBSSYSTEM
#C000	BASIC
#A000	SPIELMODULE ETC.
#9000	EIN- AUSGABE
#8000	ZEICHENGENERATOR
#2000	FREI
#1E00	VIDEO-RAM
#0400	BASIC-RAM
#0000	ZEROPAGE,STACK

2.3 3K-Erweiterung und Supererweiterung

Diese Konfiguration gleicht im Wesentlichen der vorigen.
Der Basicram beginnt hier gleich oberhalb des vom Betriebssystem beanspruchten Ram.
Die auf der vorigen Seite noch vorhandene Lücke ist hier durch eine 3K-Erweiterung aufgefüllt, d.h. daß Sie hier mit einem Anwenderspeicher von ca. 6 1/2K rechnen können.

Die Supererweiterung enthält ebenfalls eine 3K-Erweiterung, die auf die gleiche Art wirkt, wie oben beschrieben, jedoch ist hier noch eine recht nützliche Programmierhilfe zur Handhabung der hochauflösenden Grafik des VC-20 enthalten.
Diese Software belegt 4K in dem Bereich, der in der Zeichnung mit SPIELMODULE bezeichnet ist.

#E000	BETRIEBSSYSTEM
#C000	BASIC
#A000	SPIELMODULE ETC.
#9000	EIN- AUSGABE
#8000	ZEICHENGENERATOR
#6000	+ WEITERE 8K
#4000	+ WEITERE 8K
#2000	+ 8K
#1200	BASIC-RAM
#1000	VIDEO-RAM
#0400	FREI
#0000	ZEROPAGE, STACK

2.4 8K- bis 24K-Erweiterung

Die Speichereinteilung, die Sie hier sehen, tritt ein, wenn Erweiterungen ab 8K verwendet werden.
Auffällig hierbei ist, daß der Videoram ein Stück tiefer gerutscht ist, um so mit dem bereits eingebauten Ram eine lückenlose Verbindung zur Erweiterung zu haben, denn das Betriebssystem verlangt für Basic einen durchgehenden Speicher.
Hieraus folgt auch, daß eine evtl. ebenfalls eingesteckte 3K-Erweiterung keine Wirkung mehr zeigt.
Diese kann nun noch zur Ablage von Maschinenprogrammen oder einzelner Bytes mittels POKE benutzt werden.
Das Farbram hat sich ebenfalls verschoben, was Sie aber auf der nächsten Seite deutlicher sehen.

Adresse	Bereich
#9C00	I/O - 3
#9800	I/O - 2
#9600	FARBRAM B. ERW. <8K
#9400	FARBRAM B. ERW. >3K
#9120	TASTATUR
#9110	USERP.,JOYST.,IEC
#9000	VIDEOCONTROLLER

2.5 Die Ein- Ausgabebereiche

Hier finden Sie nun eine detailliertere Darstellung des auf den vorigen Seiten so salopp mit EIN- AUSGABE bezeichneten Bereichs.
Beachten Sie bitte als Besonderheit die beiden möglichen Positionen des Farbram.
Sie sind direkt abhängig von der Lage des Videoram und damit indirekt auch von der Größe des Speicherausbaus.
Bei den Bereichen I/O2 und I/O3 handelt es sich um 1K-Bereiche, die sowohl zum Anschluß von Interfaces als auch zur Speichererweiterung (allerdings nur für Maschinensprache) benutzt werden können.
Da uns bisher keinerlei Erweiterungen bekannt sind, die diese Bereiche nutzen, haben wir hier unser IEC-Bus-Modul angesiedelt, welches auf diese Weise anderen Modulen nicht den Raum nimmt.

2.6 32K und mehr

Aus dem im Abschnitt 2.1 Gesagten geht hervor, daß für eine Nutzung als Basicspeicher bei unmodifiziertem Betriebssystem nur max. 28K in Frage kommen.
In welchem Adressraum wird jedoch eine Speichererweiterung, die über 24K hinausgeht, untergebracht?
Nun, bei einer Erweiterung um zusätzliche 8K bietet sich der Bereich an, der eigentlich für Spielmodule und dergleichen vorgesehen ist. Man kann diesen Adressraum, es handelt sich um den Bereich von $A000(40960) bis $BFFF(49051), ohne weiteres mit Ram bestücken. Eines sei hier nocheinmal ganz deutlich gesagt: Erwarten Sie nicht, daß sich nun nach dem Einschalten das Betriebssystem mit mehr BYTES FREE als zuvor meldet. Das Betriebssystem benötigt nämlich als Basicspeicher einen zusammenhängenden Bereich, was hier nicht mehr der Fall ist, da der Spielebereich durch Zeichengenerator und Ein- Ausgabe vom darunterliegenden Speicher getrennt ist.
Trotzdem kann eine Speichererweiterung in diesem Block sinnvoll sein, nämlich dann, wenn Ihre Programme nicht nur aus reinem Basic, sondern auch aus Routinen, die in Maschinensprache geschrieben sind, besteht.
Solche Maschinenprogramme können Sie in eben diesem Bereich ablegen. Sie vermeiden dadurch, daß der evtl. zu knappe Basicram durch Ihre Routinen weiter reduziert wird.
Denkbar ist auch eine Verwendung dieses Bereiches als 'Geheimspeicher'. Sie können dort ungehindert Werte hineinpoken, beispielsweise Resultate aus Berechnungen, die

Sie in später nachzuladenden Programmen weiterverwenden möchten. Dieser Speicher wird nämlich beim Laden neuer Programme, selbst wenn Sie nicht die Overlaytechnik anwenden, nicht gelöscht.

Eines sollten Sie jedoch dabei beachten:
Lassen Sie eine solche Technik der Programmierung nicht zur Gewohnheit werden, d.h. daß sie die beschriebenen Verfahren selbst dann anwenden, wenn keine Notwendigkeit dazu besteht. Es könnte ohne weiteres vorkommen, daß Sie sich zum Beispiel eine Programmierhilfe oder dergleichen zulegen, die in eben diesem Bereich arbeitet, wozu auch die Supererweiterung zählt. Wollen Sie dann Ihre Programme mit einer solchen Erweiterung betreiben, so laufen sie nicht mehr, da ja an der Stelle, wo Ihr Programm Ram erwartet, sich keiner mehr befindet.

Was hat es nun mit Speichererweiterungen auf sich, die 64K und mehr beinhalten?
Wie aus dem in den vorangegangenen Abschnitten Gesagten hervorgeht, ist es zunächst schon rein physikalisch nicht möglich, mehr als 32K hinzuzufügen, da ja der Rest des Adressraumes von Betriebssystem, Basicinterpreter usw. beansprucht wird, abgesehen davon, daß das Betriebssystem nicht in der Lage ist, einen größeren Speicher zu handhaben. Da nun aber diese Grenzen vorgegeben sind, hilft man sich bei großen Erweiterungen folgendermaßen:
Man geht nicht davon aus, daß z.B. 64K Ram im Zusammenhang benutzt werden. Deshalb stückelt man die 64K oder mehr in kleinere Portionen, z.B. 8K.

Diese mundgerechten Happen kann man nun durch geeignete Vorrichtungen alternativ dem Rechner anbieten. Dieser Vorgang wird auch 'Multiplexen' genannt.
Der Rechner 'sieht' natürlich nach wie vor nur seine beispielsweise 32K Ram, allerdings kann der Anwender bestimmen, welcher Ausschnitt aus der großen Erweiterung dem Rechner angeboten werden soll.
Dieses Multiplexen kann je nach Ausführung mehr oder weniger komfortabel vor sich gehen. Eine mögliche Nutzanwendung sehr großer Speichererweiterungen könnte beispielsweise darin bestehen, daß in den einzelnen Speichersegmenten mehrere Basicprogramme vorhanden sind, die man durch entsprechendes Umschalten dem Rechner zugänglich macht. Das würde bei häufigem Programmwechsel innerhalb einer Problemstellung das lästige Nachladen erübrigen.
Grundsätzlich sollten Sie sich jedoch überlegen, ob Sie zu solchen Mitteln greifen wollen, denn Programme, die mit einer derartigen Ausstattung rechnen, sind nicht mehr transportabel, d.h. sie würden ausschließlich auf gleichartig konfigurierten Systemen laufen.

2.7 Die Modulbox VC-1020 und ihre Anwendung

Die Modulbox VC-1020 verfügt über sechs gleichwertige Slots (Steckplätze). Gleichwertig heißt, daß Sie ein beliebiges Modul in einen beliebigen Slot stecken können.
Natürlich verleitet das reichliche Vorhandensein von Slots

dazu, die Modulbox mit allen Modulen, deren man habhaft werden kann, zu bestücken. Daß Sie dabei mit überlegung vorgehen, wollen wir Ihnen in diesem Abschnitt nahebringen.

Zunächst bietet sich die Modulbox an, Ihren Rechner mit Ram-Modulen aufzustocken. Das ist auch problemlos möglich, wenn Sie beachten, daß die 8K-Module einen Sonderstatus einnehmen, und zwar dergestalt, daß sich im Inneren des Modulgehäuses eine von 1-4 numerierte Schalteranordnung befindet.

Falls Sie mehr als ein Speichermodul besitzen, dienen diese Schalter dazu, den Speicherblock zu bestimmen, in welchem das Modul wirksam werden soll.

Hier nun die Zuordnung der Schalter zu den Adressbereichen:

```
Schalter 4=EIN  $2000-$3FFF
         3=EIN  $4000-$5FFF
         2=EIN  $6000-$7FFF
         1=EIN  $A000-$BFFF
```

Von diesen Schaltern darf immer nur einer eingeschaltet sein, da sich sonst das Modul auf mehreren Adressbereichen gleichzeitig angesprochen fühlen würde.

Zwei Beispiele sollen das Gesagte verdeutlichen.

Sie besitzen drei 8K-Module. Im ersten sollte der Schalter 4, im zweiten Schalter 3 und im dritten Schalter 2 in Stellung ON sein. Alle anderen Schalter müssen in Stellung OFF stehen.

Sie haben ein 16K-Modul und ein 8K-Modul. Im 8K-Modul muß der Schalter 2=ON sein. Das 16K-Modul verfügt über keine

Schalter.
Sehen Sie hierzu auch die Abbildung in 2.4.
Bei Schalter 1=ON haben Sie den Ram im Spielebereich liegen.
Beachten Sie dazu bitte auch den Abschnitt 2.6.

Handelt es sich bei den Modulen um andere als reine Speichererweiterungen, so lassen sich solche nicht ohne weiteres miteinander betreiben.
Sie müssen nämlich zunächst feststellen, für welchen Adressbereich diese Module vorgesehen sind, da sich mehrere Module im gleichen Adressraum nicht vertragen.
Um Ihnen die Arbeit ein wenig zu erleichtern, hier eine übersicht über die Adressräume und der dafür vorgesehenen Module, soweit bekannt:

$0400-$0FFF 3K-Ram oder Ram aus der Supererweiterung
$2000-$3FFF 8K-Ram oder erste Hälfte aus 16K-Ram
$4000-$5FFF 8K-Ram oder zweite Hälfte aus 16K-Ram
$6000-$7FFF (Maschinensprache-Monitor und Programmers Aid) oder 8K-Ram
$9800-$9FFF DATA BECKER IEC-Bus
$A000-$BFFF (Superexpander und Commodore IEC-Bus) oder MAXI oder 8K-Ram oder Spielmodul

In der obigen Aufstellung sollten Sie 'oder' bitte als exklusiv-oder verstehen, d.h. das Vorhandensein einer Möglichkeit schließt eine weitere für denselben Adressraum aus.
Die Möglichkeiten der Zusammenstellung können hier natürlich nur rein hardwaremäßig betrachtet werden. über die

softwaremäßige Verträglichkeit bestimmter Module untereinander (dies trifft nicht für reine Speichererweiterungen zu) kann keine Aussage gemacht werden. Eine mögliche Bestückung der Modulbox könnte also so aussehen:

16K-Ram-Modul
Maschinensprache-Monitor
Programmers Aid
DATA BECKER IEC-Bus
MAXI

Wir hoffen, daß wir mit diesem Kapitel 2 Ihnen anschaulich vor Augen geführt haben, wie der Speicherraum des VC-20 sinnvoll zu nutzen ist, und daß Sie beurteilen können, was geht und was unmöglich ist.

3) Die VC-20 Grafik

3.1 Grundlagen

Daß der VC-20 bereits in der Grundversion beachtliche Grafikmöglichkeiten besitzt, hat zwar jeder bereits einmal irgendwo gehört, doch wie man diese Grafik erzeugen kann, ist für die Meisten bisher ein Geheimnis geblieben. Im VC-20 Handbuch finden Sie diese Möglichkeiten des VC-20 gar nicht erwähnt. In diesem Kapitel sagen wir Ihnen, was Sie über das Videochip des VC-20 wissen müssen, und wie sie mit diesem Wissen hochauflösende Grafiken erzeugen können. Sie werden imstande sein, auf der Grundversion des VC-20 Grafiken mit einer Auflösung von 128*128 Punkten und mit Hilfe der normalen 8- bzw. 16K-Erweiterung Grafiken mit einer Auflösung von 168*176 Punkten zu erzeugen. All dies werden Sie anhand von Beispielen und lauffähigen Programmen, die Sie nur noch abtippen müssen, erlernen.
Zuerst wenden wir uns dem Video-Chip des VC-20 zu. Dieser hochintegrierte Baustein mit der Nummer 6560 ist speziell für kleine Computer und Video Spiele entwickelt worden. Mit Hilfe seiner 15 Register können Sie alle Betriebsarten des VC-20 kontrollieren. Von diesem Chip werden nicht nur die Grafik und die Farben kontrolliert, sondern es enthält auch die Tongeneratoren und die AD-Wandler für die Paddles. Auf der nächsten Seite finden Sie eine übersicht über alle Register dieses Chips, dann folgt eine Erklärung der einzelnen Register.

		7	6	5	4	3	2	1	0	Bitnummer
CR0	9000	I	S6	S5	S4	S3	S2	S1	S0	Bildschirmanfang X-Koordinate
KR1	9001	S7	S6	S5	S4	S3	S2	S1	S0	Bildschirmanfang Y-Koordinate
KR2	9002	V9	M6	M5	M4	M3	M2	M1	M0	Anzahl der Spalten auf dem Schirm
KR3	9003	R0	N5	N4	N3	N2	N2	N0	D	Anzahl der Reihen auf dem Schirm
KR4	9004	R8	R7	R6	R5	R4	R3	R2	R1	Rasterwert
KR5	9005	V13	V12	V11	V10	C13	C12	C11	C10	Start Schirm + Charactergenerator
KR6	9006	LH7	LH6	LH5	LH4	LH3	LH2	LH1	LH0	Lichtgriffelposition horizontal
KR7	9007	LV7	LV6	LV5	LV4	LV3	LV2	LV1	LV0	Lichtgriffelposition vertikal
KR8	9008	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0	Potentiometerposition horizontal
KR9	9009	PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0	Potentiometerposition vertikal
KRa	900a	S1	F16	F15	F14	F13	F12	F11	F10	Frequenz Tongenerator 1
KRb	900b	S2	F26	F25	F24	F23	F22	F21	F20	Frequenz Tongenerator 2
KRc	900c	S3	F36	F35	F34	F33	F32	F31	F30	Frequenz Tongenerator 3
KRd	900d	S4	F46	F45	F44	F43	F42	F41	F40	Frequenz Tongenerator Rauschen
KRe	900e	FA3	FA2	FA1	FA0	A3	A2	A1	A0	Lautstärke + Hilfsfarbe
KRf	900f	FB3	FB2	FB1	FB0	R	FC2	FC1	FC0	Farben

Beschreibung der Register :

In Spalte 1 finden Sie die Bezeichnung des Registers, in Spalte 2 die Speicherstelle im RAM, wo dieses Register zu finden ist. Diese Zahlen sind im Hexadezimalsystem. Das erste Register befindet sich also in der Speicherstelle 9000 im Hexadezimalsystem bzw. 36864 im Dezimalsystem. Die weiteren Register befinden sich in den folgenden Speicherstellen.

Erklärung der einzelnen Register :

KR 0 : Die Bits 0-6 bestimmen, wieviele Punkte das Bild vom linken Rand des Schirms entfernt ist. Es wird benutzt, um das Bild horizontal zu zentrieren. Bit 7 bestimmt die Zwischenzeilenabtastung. Gewöhnlich steht hier eine 0.

KR 1 : Dieses Register bestimmt, wieviele Punkte vom oberen Bildschirmrand das Bild beginnt. Es wird benutzt, um das Bild vertikal zu zentrieren.

KR 2 : Die Bits 0-6 bestimmen die Anzahl der Spalten auf dem Bildschirm, d.h. wieviele Buchstaben pro Reihe vorhanden sind. Bit 7 entspricht Bit 9 der Anfangsadresse der Video-Matrix und gehört zu den Bits 4-7 des Registers 5.

KR 3 : Die Bits 1-6 bestimmen die Anzahl der Bildschirmzeilen. Mit Bit 0 wird festgelegt, ob die Buchstaben in einer 8*8 (D=0) oder in einer 8*16 (D=1) Matrix dargestellt werden. Die Darstellung der Buchstaben in einer 8*16 Matrix ist hauptsächlich für hochauflösende

Grafiken gedacht. Wir werden davon Gebrauch machen, wenn wir die Grafik mit der Auflösung von 168*176 Punkten erstellen. Bit 7 ist Teil des Wertes in Register 4.

KR 4 : Dieses Register enthält die Nummer der Zeile, die gerade von dem Rasterstrahl bearbeitet wird.

KR 5 : Die Bits 0-3 sind die Bits 10-13 der Startadresse für den Charactergenerator. Die Bits 4-7 bilden zusammen mit Bit 7 aus Register 2 die Bits 10 bis 13 der Startadresse des Video-RAM.

KR 6 : Dieses Register enthält die horizontale Position des Lichtgriffel.

KR 7 : Dieses Register enthält die vertikale Position des Lichtgriffels.

KR 8 : Dieses Register enthält den digitalisierten Wert des Potentiometers 0.

KR 9 : Dieses Register enthält den digitalisierten Wert des Potentiometers 1.

KR a : Die Bits 0-6 bestimmen die Frequenz des Tongenerators 1, Bit 7 schaltet diesen Tongenerator ein (S=1).

KR b : Die Funktion dieses Registers entpricht der des Registers a, nur für Tongenerator 2.

KR c : Die Funktion dieses Registers entspricht der des Registers a, nur für Tongenerator 3.

KR d : Die Funktion dieses Registers entspricht der des Registers a, nur für den Rauschgenerator.

KR e : Die Bits 0-3 bestimmen die Gesamtlautstärke der Tongeneratoren. Die Bits 4-7 bestimmen die Hilfsfarbe, wenn der "Vielfarb"-Betriebsmodus eingeschaltet ist.

KR f : Die Bits 0-2 bestimmen eine der 8 Farben für den Bildschirmrahmen. Die Bits 4 - 7 bestimmen die Hintergrundfarbe des Bildschirms. Das Invertierungsbit R bestimmt, ob verschiedenfarbene Buchstaben auf einem einheitlichen Hintergrund (R=1) dargestellt werden, oder ob die Darstellung invertiert wird (R=0), d.h. alle Buchstaben haben die gleiche Farbe (bestimmt durch die Bits 4-7 des f-Registers) und der Hintergrund jedes Buchstaben vom Wert des Farb-RAM bestimmt wird.

Wenn Sie die vorangegangenen Seiten aufmerksam durchgelesen haben, werden Sie festgestellt haben, daß es keinen eigentlichen hochauflösenden Grafikmodus gibt. Sie werden aber auch festgestellt haben, daß sich sowohl das Video-RAM als auch der Charactergenerator verschieben lassen. Das verantwortliche Register ist hier das Kontrollregister 5. Außerdem gibt es die Möglichkeit, die Zeichen in einer 8*16 Matrix darzustellen. Wir werden nun zuerst ein bißchen mit dem Charactergenerator spielen und Ihnen anschließend zeigen, wie sie so hochauflösende Grafiken erzeugen können.

3.2 Programmierbare Zeichen

Der Charactergenerator beginnt normalerweise ab der Speicherstelle 32768. Mit Hilfe des Kontrollregisters 5 können wir nun die Startadresse in den RAM-Bereich verlegen. Die folgenden Tabelle gibt Ihnen Auskunft, welche Werte Sie in dieses Register schreiben müssen, um den Charactergenerator an eine bestimmte Stelle zu verschieben.

Inhalt KR5	Speicherstelle	Inhalt der Speicherstelle
240	32768	Charactergenerator ROM
241	33792	Charactergenerator ROM
242	34816	Charactergenerator ROM
243	35840	Charactergenerator ROM
244	(36864)	Videochip
245	(37888)	Farb-RAM
246	(38912)	nicht wählbar
247	(39936)	nicht wählbar
248	(0)	Null-Seite
249	(1024)	RAM bei Speichererweiterung
250	(2048)	RAM bei Speichererweiterung
251	(3192)	RAM bei Speichererweiterung
252	4096	BASIC-Start
253	5120	RAM
254	6144	RAM
255	7168	RAM

Der normale Inhalt des Kontrollregisters 5 ist entweder 240 bei Großbuchstaben mit Graphik oder 242 bei Klein- und Großbuchstaben. Bei dieser Tabelle liegt das VideoRAM ab Speicherstelle 7680.

Nehmen Sie nun alle Speichererweiterungen aus Ihrem VC-20 und tippen Sie das folgende kleine Programm ein:

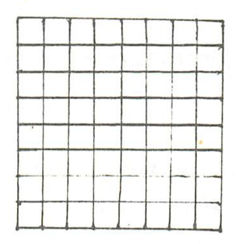
```
 5 POKE 52,20:POKE 56,20:CLR
10 FORI=0TO2047:POKE5120+I,PEEK(32768+I):NEXTI
20 POKE 36869,253
```

In Zeile 5 setzen wir den für BASIC verfügbaren Speicherplatz herunter, um den Charactergenerator vor dem überschreiben zu schützen. In Zeile 10 kopieren wir den Charactergenerator aus dem ROM in den RAM-Bereich ab Speicherstelle 5120. Mit Zeile 20 schließlich versetzen wir den Zeiger auf den Zeichengenerator auf die Speicherstelle 5120. Wenn Sie nun Buchstaben eintippen, stellen Sie fest, daß sich nichts verändert hat. Da wir bis jetzt auch nur den Charactergenerator kopiert haben, ist dies auch korrekt. Tippen Sie nun die folgenden Zeilen ein:

```
30 FORI=0TO7:READA:POKE5128+I,A:NEXT
100 DATA 255,129,129,129,129,129,129,255
```

Lassen Sie das Programm laufen. Sofort verändern sich alle "A"s auf dem Bildschirm zu kleinen Quadraten. Wenn Sie nun ein "A" eintippen, so erhalten Sie jetzt immer ein kleines Quadrat auf dem Bildschirm. Beachten Sie aber, daß dieses kleine Quadrat immer noch als "A" funktioniert. Befehle wie "LOAD" oder "SAVE" mögen jetzt vielleicht etwas komisch aussehen, funktionieren aber weiterhin.

Wie funktioniert nun der Zeichengenerator? Alle Buchstaben und graphischen Zeichen des VC-20, die auch auf der Tastatur abgebildet sind, werden aus einer 8*8 Matrix aufgebaut. Eine solche Matrix sieht also folgendermaßen aus:



Stellen Sie sich nun vor, jede Reihe entspricht einem Byte, also einem Speicherplatz, und jeder der 8 Punkte einem Bit dieses Bytes. Das linke Bit soll die größte Wertigkeit haben. Von links beginnend, haben die 8 Stellen die Wertigkeiten 128,64,32,16,8,4,2,1 . Wenn nun ein Punkt gesetzt ist, so wird der entsprechende Wert in die Speicherstelle addiert. Dazu wieder ein Beispiel :

0	1	1	1	1	1	0	0
0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0
0	0	1	1	1	1	0	0
0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0
0	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0

Diese Matrix entspricht dem Buchstaben B. Die Werte der Speicherstellen von oben nach unten sind 124,34,34,60,34,34, 124,0. Tippen Sie nun die folgende Zeile ein:

```
FORI=5136TO5143:PRINTPEEK(I):NEXT
```

Sie sehen, sie erhalten die gleichen Werte.
Fassen wir zusammen: Alle Zeichen werden in einer 8*8-Matrix dargestellt. Die 8 Reihen werden im Charactergenerator hintereinander abgespeichert.
Sie sehen, daß es nun für Sie ein leichtes ist, Ihren Zeichensatz zu verändern, zum Beispiel könnten Sie Ihrem VC-20 die deutschen Umlaute ä,ö,ü und ß beibringen. Doch wie kommen wir nun zu der versprochenen hochauflösenden Grafik?
Vielleicht ahnen Sie die Antwort bereits. Ganz einfach, wir schreiben den Bildschirm mit verschiedenen Zeichen voll und programmieren diese so, wie wir sie benötigen. Da wir nur 256 verschiedene Zeichen haben, können wir leider nicht den gesamten Bildschirm ausnutzen. Wir können pro Reihe 16 Zeichen unterbringen. Bei einer Auflösung von 8 Punkten pro Zeichen ergibt sich, daß wir 8*16=128 Punkte ansteuern können. Das Gleiche gilt für die vertikale Richtung. Wir benötigen demnach 2K Speicherplatz für die Grafikseite, so daß uns bei der Grundversion des VC-20 noch 1,5 K für Programme zur Verfügung stehen. Sie werden nun sicher sagen, dies sei nicht viel. Daß sich aber auch in diesem Speicherplatz bereits vernünftige Programme unterbringen lassen, zeigen wir Ihnen mit dem folgenden Programm. Es ist ein Funktionenplotter, mit dessen Hilfe Sie alle Funktionen zeichnen können. Tippen Sie nun das folgende Programm ein:

```
5 POKE52,20:POKE56,20:CLR
10 PRINT"♥"
20 PRINT"GEBEN SIE DEN STARTPUNKT EIN ":INPUTS
30 PRINT"GEBEN SIE DEN ENDPUNKT EIN ":INPUTE
40 PRINT"SOLL DAS ALTE BILD GELOESCHT WERDEN (J/N)
50 GET A$:IFA$=""THEN50
70 PRINT"♥"
80 POKE36869,253:POKE36879,8
84 PRINT"♥":FORR=0TO15:FORL=0TO15
90 POKE7680+R*22+L,R*16+L
100 NEXTL,R
101 POKE36869,253:POKE36879,8
102 IFA$<>"J"THEN130
103 FORI=5120TO7168:POKEI,0:NEXT
104 FORU=0TO127:Y=64:GOSUB1000:NEXT
105 FORY=0TO127:U=64:GOSUB1000:NEXT
130 U=0:FORX=STOESTEP(E-S)/127
135 IFX=0THEN170
140 GOSUB2000
150 Y=64-Y*64
160 GOSUB1000
170 U=U+1
180 NEXTX
190 GETA$:IFA$=""THEN190
200 GOTO200
210 END
1000 AD=5120
1010 AY=INT(Y/8)*128+(YAND7)
1020 AD=AD+8*INT(U/8)+AY
1030 MA=2↑(7-(UAND7))
1040 POKEAD,PEEK(AD)ORMA
1050 RETURN
2000 Y=COS(X)
2010 RETURN

READY.
```

Erläuterungen zu diesem Programm:

Zeile 5 :	Der für BASIC verfügbare Speicherplatz wird begrenzt
Zeile 90:	Der Bildschirmspeicher wird mit den Zeichen beschrieben.
Zeile 101:	Der Zeiger auf den Zeichengenerator und die Farbe werden gesetzt.
Zeile 103:	Der Charactergenerator wird gelöscht, d.h. der Bildschirm leert sich.
Zeile 104:	Die X-Achse wird gezeichnet.
Zeile 105:	Die Y-Achse wird gezeichnet.
Zeile 130-180:	Die Funktion wird gezeichnet.
Zeile 1000-1050:	Die X und Y-Werte werden umgerechnet und die Punkte mit der Anweisung in Zeile 1040 gezeichnet.
Zeile 2000:	Hier steht die Funktion, die gezeichnet werden soll.

Lassen Sie das Programm nun laufen. Wenn Sie als Anfangspunkt -3.14 und als Endpunkt 3.14 wählen und die Frage nach dem Löschen mit "J" beantworten, werden Sie eine schöne Kosinuskurve auf dem Bildschirm sehen.

Wenn das Bild fertig ist, brechen Sie die Endlosschleife in Zeile 200 durch Drücken der RUN/STOP und RESTORE Taste ab. Ändern Sie nun Zeile 2000, schreiben Sie dort zum Beispiel die Funktion Y=SIN(X)/X hinein. Starten Sie nun das Programm, geben Sie als Grenzen -15 und 15 ein und beantworten Sie die Fragen nach dem Löschen mit "N". Sie sehen, daß die alte Kurve noch vorhanden ist. Die neue Kurve wird nun einfach dazugezeichnet. Dies können Sie mit immer wieder neuen Funktionen so oft wiederholen, wie Sie möchten.

3.3 Laufschrift

Als weitere Anwendung des Charactergenerators möchten wir Ihnen nun noch zeigen, wie Sie riesengroße Buchstaben auf dem Bildschirm und damit eine weithin lesbare Laufschrift erzeugen können.

Das Prinzip ist ganz einfach:

Sie speichern einen Text in eine Variable. Diesen Text gehen Sie Buchstabe für Buchstabe durch und holen sich den entsprechenden Code aus dem Charactergenerator. Für jedes gesetzte Bit schreiben Sie nun je nach Vergrößerung einen oder zwei Zeichen auf dem Bildschirm. Dann verschieben Sie den ganzen Bildschirm nach links und nehmen sich den nächsten Buchstaben vor. Sind Sie am Ende Ihres Textes angelangt, fangen Sie wieder von vorne an. Auf diese Art und Weise erhalten Sie praktisch eine sich endlos wiederholende Schriftrolle.

Auf der nächsten Seite finden Sie das Listing zu einem Laufschriftprogramm. Es kann einen Text, der bis zu 255 Zeichen lang sein darf, in 4 verschiedenen Größen über den Bildschirm laufen lassen. Das Programm unterteilt sich in fünf Teile. Im ersten Teil bis Zeile 90 werden die Anfangsvariablen gesetzt, der Text eingelesen und der Vergrößerungsfaktor gewählt. Entsprechend diesem Faktor wird nun in einen der vier nachfolgenden Programmteile verzweigt. Ihnen wird auffallen, daß die Programmteile 3 und 4 fast identisch sind. Aus Gründen der übersichtlichkeit haben wir darauf verzichtet, diese in einen Teil zusammenzufassen.

Auf der nächsten Seite nun das Listing, danach noch Erläuterungen zu den wichtigsten Programmzeilen.

```
0 REM **** LAUFSCHRIFT ****
1 POKE56,24:POKE52,24:CLR
2 VA=4096:BB=22:ZZ=22:FA=37888:CG=32768
3 FORI=6144TO7417STEP2
4 POKEI,PEEK(CG):POKEI+1,PEEK(CG):CG=CG+1:NEXTI
5 CG=32768:POKE36879,8:PRINT"■":REM "■" ENTSPRICHT CTRL/2 (FARBE WEISS)
6 DIM W$(15)
10 PRINTCHR$(147)
20 PRINT"GEBEN SIE IHREN TEXT EIN":INPUTA$
30 PRINT"GEBEN SIE DEN VERGROESERUNSFAKTOR EIN :":INPUTV
40 IFV<1ANDV>4THEN30
50 PRINTCHR$(147)
55 X=0:Y=ZZ/2
90 ONVGOTO100,1000,2000,3000
100 FORJ=1TOZZ/2+1:PRINT:NEXT
105 W$=A$:IFLEN(A$)>BBTHENW$=LEFT$(W$,BB):A$=RIGHT$(A$,LEN(A$)-BB)+" ":GOTO110
106 IFLEN(A$)<BBTHENFORQ=1TOBB-LEN(W$):W$=W$+" ":NEXTQ:A$=RIGHT$(A$,BB-LEN(W$))+
" "
110 Z$=LEFT$(W$,1):W$=RIGHT$(W$,BB-1)+LEFT$(A$,1)
120 A$=RIGHT$(A$,LEN(A$)-1)+Z$
130 PRINTCHR$(145);CHR$(145);W$
135 FORW=QTO100:NEXT
140 GOTO110
1000 FORJ=0TO7:FORI=0TOBB:W$(J)=W$(J)+" ":NEXTI,J
1005 PRINTCHR$(147);:FORI=1TO7:PRINT:NEXT
1010 FORI=1TOLEN(A$)
1020 Z$=MID$(A$,I,1)
1030 ZZ=CG+(ASC(Z$)-64)*8
1040 IFASC(Z$)=32THENZZ=6654
1050 FORBI=7TO0STEP-1:FORR=0TO7:Z=PEEK(ZZ+R):Z=ZAND2↑BI
1070 IFZ=2↑BITHENW$(R)=RIGHT$(W$(R),BB-1)+"▒":GOTO1090
1080 W$(R)=RIGHT$(W$(R),BB-1)+" "
1090 NEXTR
1300 FORJ=0TO7:PRINTW$(J);:NEXTJ
1350 FORP=1TO8:PRINTCHR$(145);:NEXTP
```

```
1400 NEXTBI
1500 NEXTI
1600 GOTO1010
2000 FORJ=0TO15:FORI=0TOBB:W$(J)=W$(J)+"  ":NEXTI,J
2005 PRINTCHR$(147):PRINT:PRINT:
2010 FORI=1TOLEN(A$)
2020 Z$=MID$(A$,I,1)
2030 ZZ=CG+(ASC(Z$)-64)*8
2040 IFASC(Z$)=32THENZZ=6654
2050 FORBI=7TO0STEP-1:FORR=0TO15:Z=PEEK(ZZ+INT(R/2)):Z=ZAND2↑BI
2070 IFZ=2↑BITHENW$(R)=RIGHT$(W$(R),BB-1)+"▒":R=R+1:W$(R)=RIGHT$(W$(R),BB-1)+"▒"
:GOTO1090
2080 W$(R)=RIGHT$(W$(R),BB-1)+"  ":R=R+1:W$(R)=RIGHT$(W$(R),BB-1)+"  "
2090 NEXTR
2300 FORJ=0TO15:PRINTW$(J);:NEXTJ
2350 FORP=1TO16:PRINTCHR$(145);:NEXTP
2400 NEXTBI
2500 NEXTI
2600 GOTO2010
3000 POKE 36865,19
3010 POKE 36879,47
3020 POKE 36867,89
3030 POKE 36869,254
3040 FORJ=1TO9:PRINT:NEXT
3050 GOTO105

READY.
```

Teil 2 dieses Programms geht von Zeile 100 bis Zeile 140. In diesen Programmteile kommen Sie, wenn Sie den Vergrößerungsfaktor 1 wählen, d.h. der Text erscheint in Originalgröße. In Zeile 100 wird der Cursor auf die mittlere Bildschirmzeile gesetzt. Wenn der Text länger als die Bildschirmzeile ist, wird in Zeile 105 der Text in zwei Teile gespalten. Die Variable W$ enthält den linken Teil des Textes, entsprechend der Breite des Bildschirms. Die Variabel A$ enthält den restlichen Text. Ist die Länge des Textes kleiner als die Breite des Bildschirms, so wird die Variable W$ mit soviel Leerzeichen aufgefüllt, bis die Länge von W$ der Zeilenbreite entspricht. Der Variablen A$ wird nun ein Leerzeichen hinzugefügt. In Zeile 110 wird der linke Buchstabe der Variablen W$ der Variablen Z$ zugewiesen. Dieser Buchstabe wird nun aus W$ entfernt und der linke Buchstabe aus A$ hinzugefügt. In Zeile 120 nun wird der linke Buchstabe aus A$ entfernt und Z$ hinzugefügt. Dann wird W$ ausgedruckt, in Zeile 135 eine Warteschleife abgearbeitet und danach wieder weiter geschoben. Sie sehen, daß die eigentliche Verschiebung nur aus den Zeilen 110 und 120 besteht. Sollte die Laufschrift Ihnen zu langsam oder zu schnell sein, so können Sie Zeile 135 entsprechend verändern.

Teil 3 und 4 des Programms sind wie schon erwähnt fast identisch. Wir wollen hier deshalb nur Teil 3 erklären. In Zeile 1000 werden die Variablen W$(J) mit soviel Leerzeichen gefüllt, wie der Bildschirm breit ist. Der PRINT-Befehl in Zeile 1005 bewirkt ein Löschen des Bildschirms und die Positionierung des Cursors auf der mittleren Zeile. Zeile 1010 startet eine Schleife, die alle Buchstaben des Textes in der Variablen A$ durchgeht. In Zeile 1020 wird der laufende Buchstabe der Variablen Z$ zugewiesen und in Zeile 1030 die Position im Charactergenerator errechnet, wo die Matrix dieses Buchstabens abgespeichert ist. Die Formel dafür lautet Startposition des Charactergenerators (CG) plus 8 mal

der Differenz aus ASCII-Code und der Zahl 64. In ZZ steht nun also die Startposition der Matrix für den Buchstaben. Eine Ausnahme bildet lediglich das Leerzeichen. Es wird deshalb in Zeile 1040 gesondert behandelt. Beachten Sie , daß sie nur Großbuchstaben verwenden dürfen. Diese Matrix wird nun spaltenweise abgearbeitet. Zuerst wird das höchste Bit der ersten Reihe der Variablen Z zugewiesen. Ist dieses Bit gesetzt, so wird in Zeile 1070 ein Graphikzeichen der Variablen W$(J) zugefügt und gleichzeitig das linke Zeichen entfernt. Ist dieses Bit nicht gesetzt, so wird ein Leerzeichen hinzugefügt. Das Gleiche geschieht nun mit dem höchsten Bit der zweiten Reihe u.s.w. Danach wird das zweithöchste Bit bei allen 8 Reihen bearbeitet, dann das dritthöchste bis zum niedrigsten. Anschließend wird der nächste Buchstabe bearbeitet und wenn der ganze Text auf dem Bildschirm erschienen ist, fängt das Programm wieder beim ersten Buchstaben an.

Im Teil 4 ist der einzige Unterschied zu Teil 3 die Schleife mit der Variablen R. Diese Schleife läuft hier doppelt so lange. Das Anfügen der Buchstaben bzw. der Leerzeichen geschieht hier für jedes Bit zweimal. Dadurch ergibt sich, daß die Buchstaben doppelt so groß werden wie in Teil 3.

In Teil 5 schließlich ab Zeile 3020 wird die Darstellung der Zeichen etwas verändert. Der POKE-Befehl in Zeile 3020 bewirkt, daß die Zeichen in doppelter Größe dargestellt werden. Dazu gehören auch noch die Zeilen 1,4 und 5 . In Zeile 1 wird der für BASIC verfügbare Speicherplatz heruntergesetzt. In den Zeilen 4 bis 5 wird der Charactergenerator ins RAM kopiert, wobei er derart verändert wird, daß jede Reihe eines Zeichen zweimal hintereinander gespeichert wird. In Zeile 3030 wird der Zeiger auf den Charactergenerator umgesetzt. Die Zeilen 3000 und 3010 bewirken die Formatierung des Bildschirms und die Auswahl der Farbe.

3.4 Grafik mit Speichererweiterung

Für die folgenden Programme benötigen Sie mindestens eine 8K-Speichererweiterung. Sie werden in diesem Kapitel zuerst eine Grafik mit einer Auflösung von 168*176 Punkten kennenlernen, anschließend zeigen wir Ihnen ein Maschinenspracheprogramm , das die Handhabung der Grafik erheblich beschleunigt, dann einen etwas komfortableren Funktionenplotter und noch einige Grafiken, die einfach schön anzusehen sind.
Doch zuerst müssen wir auch hier wieder etwas Theorie einschieben. Im Prinzip werden wir jetzt wieder das Gleiche machen wie ohne Speichererweiterung. Wir werden also den Charactergenerator wieder in den RAM-Bereich verschieben und den Bildschirm mit 240 verschiedenen Zeichen füllen. Sie fragen sich nun, wie die höhere Auflösung entsteht? Blättern Sie doch einmal kurz zu der Erklärung des Kontrollregisters 3 (Seite 20) zurück. Dort steht, daß es die Möglichkeit gibt, mit einem Bit auf eine Darstellung der Zeichen in einer 8*16 Matrix umzuschalten. Genau davon machen wir jetzt Gebrauch.
Stecken Sie mindestens eine 8K-Speichererweiterung in Ihren Rechner und schalten Sie das Gerät ein. Wir benötigen für die Grafikseite jetzt 4K-RAM Speicherplatz. Aus diesem Grund verlegen wir nun zuerst den BASIC-Start um 4K nach oben. Tippen Sie Folgendes ein:

```
POKE44,34:POKE34*256,0:NEW
```

Wenn Sie nun den Befehl PRINTFRE(I) eingeben, müßten 4K weniger Speicher zur Verfügung stehen als beim Einschalten.
Wir werden Ihnen nun anhand eines kleinen BASIC-Programms die Grafik zeigen, bevor wir die Sache mit Hilfe von Maschinensprache etwas beschleunigen. Tippen Sie nun das folgende Programm ein:

```
20 GOSUB10000
30 FORI=-πTOπSTEP2*π/167
40 X=U:Y=SIN(I)*88+88
50 GOSUB10200:U=U+1:NEXT
60 GETA$:IFA$=""THEN60
70 GOSUB10100:END
10000 REM UMSCHALTUNG AUF GRAFIK
10030 PRINT"♥":FORI=4352TO8047:POKEI,0:NEXT
10040 POKE36867,151:POKE36866,21:POKE36869,204
10050 POKE36864,14
10060 FORI=16TO255:POKE4096+I-16,I:POKE37888+I-16,6:NEXTI
10070 RETURN
10100 REM UMSCHALTUNG AUF TEXT
10130 POKE36864,12:POKE36866,22:POKE36867,174
10140 POKE36869,192:PRINT"♥"
10150 RETURN
10200 REM PUNKT SETZEN
10230 GOSUB10400
10240 POKEAD,PEEK(AD) OR MA
10250 RETURN
10400 AD=4352
10410 AY=INT(Y/16)*336+(YAND15)
10420 AD=AD+16*INT(X/8)+AY
10430 MA=2↑(7-(X AND 7))
10440 RETURN

READY.
```

Mit dem vorstehenden Programm können Sie eine Sinuskurve auf dem Bildschirm zeichnen.
Die Zeilen 30 bis 70 enthalten nur die übliche FOR-NEXT-Schleife zur Berechnung der Sinus-Werte. Interessant sind erstmal die Zeilen 10030 bis 10050. In Zeile 10030 wird der Charactergenerator gelöscht. Das kennen Sie bereits aus der Grafik mit der Grundversion des VC-20, nur daß der Speicherbereich hier ein anderer ist. Die vier POKE-Befehle in den folgenden beiden Zeilen müssen wir uns aber nun etwas genauer ansehen.
Halten Sie die Tabelle mit den Kontrollregistern des Videochip bereit. Wir erklären Ihnen nun die vier POKE-Befehle sehr ausführlich, da Sie für das Verständnis der Grafik ganz wichtig sind. Anhand der binären Darstellung der Werte können Sie genau erkennen, welche Bits der Register gesetzt werden.

1) POKE 36867,151   151 (DEZIMAL) = 10010111 (BINäR)
   Dieser POKE-Befehl belegt das Kontrollregister 3 mit dem Wert 151. Bit 0 ist dafür verantwortlich, daß die Darstellung in einer 8*16 Matrix gewählt wird. Die Bits 1 bis 5 bestimmen die Anzahl der Zeilen der Video Matrix, hier werden 11 Reihen festgelegt.

2) POKE 36866,21   21 (DEZIMAL) = 00010101 (BINäR)
   Dieser POKE-Befehl belegt das Kontrollregister 2 mit dem Wert 21. Dies bedeutet, daß pro Zeile 21 Zeichen zur Verfügung stehen.

3) POKE 36869,204   204 (DEZIMAL) = 11001100 (BINäR)
   Dieser POKE-Befehl belegt das Kontrollregister 5 mit dem Wert 204. Daraus resultiert für das Video-RAM und den Charactergenerator die Startadresse 4096.

4) POKE 36864,14 14 (DEZIMAL) = 00001110 (BINäR)

Dieser POKE-Befehl belegt das Kontrollregister 0 mit dem Wert 14. Dies bedeutet, daß wir den Bildschirm horizontal zentrieren.

In Zeile 10060 wird der Bildschirm mit den verschiedenen Zeichen gefüllt, wie Sie das auch schon von der Grafik in der Grundversion kennen. Gleichzeitig wird das Farb-RAM initialisiert. Im Unterprogramm ab Zeile 10400 werden die X- und Y-Werte wieder in die entsprechenden Speicherstellen umgerechnet. Beachten Sie, daß im Unterschied zur Grafik in der Grundversion die Y-Werte hier mit der Zahl 16 berechnet werden. In Zeile 10240 wird dann der eigentliche Punkt gesetzt.
In den Zeilen 10130 und 10140 wird der alte Zustand wieder hergestellt, d.h. es wird wieder in den Text-Modus geschaltet.

Fassen wir zusammen:
Wir haben auf dem VC-20 mit mindestens 8K-Speichererweiterung eine Grafik mit einer Auflösung von 168*176 Punkten erzeugt. Die Speichererweiterung ist notwendig, da allein für die Grafik bereits 4K Speicher benötigt werden. Die Grafik wird mit Hilfe der programmierbaren Zeichen erzeugt. Dreh- und Angelpunkt ist hierbei das Kontrollregister 5 des Videochip, mit dessen Hilfe der Charactergenerator in den RAM-Bereich verlegt werden kann. Wichtig ist weiterhin die Umschaltung auf die Darstellung der Zeichen in einer 8*16 Matrix, die mit Hilfe des Bits 0 im Kontrollregister 3 geschieht.

3.5 Grafikhilfe mit Maschinensprache

All dies haben wir bis jetzt von BASIC aus gemacht. Dadurch ergeben sich erhebliche Wartezeiten, besonders bei der Initialisierung der Grafik. Um dieses Manko auszuräumen, haben wir eine universelles Grafikprogramm in Maschinensprache erstellt. Das Assemblerlisting zu diesem Programm finden Sie auf den nächsten Seiten. Wenn Sie einen Monitor oder einen Assembler besitzen, können Sie es so abtippen. Damit Sie aber auch als nur BASIC-Programmierer in den Genuß dieses Programms kommen können, finden Sie nach dem Assemblerlisting ein BASIC-Programm, das in den DATA-Zeilen das Maschinenspracheprogramm enthält. Tippen Sie dieses Programm ab. Es prüft die Summe aller Zahlen, um sich zu vergewissern, daß Sie nichts falsch abgetippt haben. Bevor Sie das Programm laufen lassen, speichern Sie es unbedingt vorher ab, damit es sich bei eventuellen Tippfehlern nicht selber löscht.
Bevor Sie beginnen, tippen Sie zuerst die folgende Zeile ein:

```
POKE44,34:POKE34*256,0:NEW
```

Mit dieser Zeile wird wieder der BASIC-Start hochgesetzt, um Platz für die Grafikseite und das Maschinenspracheprogramm zu schaffen. Diese Zeile müssen Sie jedesmal eintippen, bevor Sie mit der Grafik und dem Programm arbeiten können!! Sollten Sie dies vergessen, löscht sich das Programm selbst.
Beachten Sie, daß sich Charactergenerator und VideoRAM überlappen. Die praktische Konsequenz daraus ist, daß Sie keine Eingaben machen können, ohne Ihr Bild zu zerstören. Aus diesem Grund finden Sie in den Grafikprogrammen die Abfrage zum Namen des Bildes zu Beginn des Programms.

```
100:    2000                          .OPT P1
130:    2000              XCOORD   =     $14
140:    2000              FLAG     =     $97      ; PUNKT SETZEN/LOESCHEN
150:    2000              MASKE    =     FLAG     ; MASKE FUER HARDCOPY
160:    2000              SA       =     $B9      ; SEKUNDAERADRESSE
170:    2000              TMP      =     $FD
180:    2000              ADR      =     TMP
190:    2000              AV       =     TMP
220:    2000              COLLO    =     $9400    ; ANFANG HI-RES FARBRAM
230:    2000              COLHI    =     $9600    ; ENDE      "
240:    2000              GRALO    =     $1100    ; ANFANG HI-RES BITMAP
250:    2000              GRAHI    =     $2000    ; ENDE      "
260:    2000              GETCOR   =     $D7EB    ; HOLT X- UND Y-COORDINATEN
270:    2000              CHKCOM   =     $CEFD    ; PRUEFT AUF KOMMA
280:    2000              GETBYT   =     $D79E    ; HOLT BYTE-WERT
290:    2000              VIDEO    =     $9000    ; VIDEOCONTROLLER
300:    2000              GETPAR   =     $E1D1    ; HOLT FILENAMEN UND GERAETENUMMER
310:    2000              CLRSCR   =     $E55F    ; LOESCHT BILDSCHIRM
340:    2000              LOAD     =     $FFD5    ; LOAD ROUTINE
350:    2000              SAVE     =     $FFD8    ; SAVE ROUTINE
                           ;
370:    2000                       *=    $2000    ; SPRUNGTABELLE FUER FUNKTIONEN
380:    2000 4C 1B 20              JMP   INIT     ; GRAFIK-MODUS EINSCHALTEN
390:    2003 4C 3E 20              JMP   CLEAR    ; GRAFIK LOESCHEN
400:    2006 4C 55 20              JMP   COLOR    ; FARBE SETZEN
410:    2009 4C 71 20              JMP   REVERS   ; GRAFIK INVERTIEREN
420:    200C 4C 8D 20              JMP   SET      ; GRAFIKPUNKT SETZEN
430:    200F 4C 8A 20              JMP   RESET    ; GRAFIKPUNKT LOESCHEN
440:    2012 4C 45 21              JMP   GLOAD    ; GRAFIK LADEN
450:    2015 4C 2C 21              JMP   GSAVE    ; GRAFIK ABSPEICHERN
470:    2018 4C 54 21              JMP   GOFF     ; GRAFIK AUS
                           ;
490:    201B A9 97        INIT     LDA   #151     ; GRAFIK EIN
500:    201D 8D 03 90              STA   36867
510:    2020 A9 15                 LDA   #21
510:    2022 8D 02 90              STA   36866
520:    2025 A9 CC                 LDA   #204
520:    2027 8D 05 90              STA   36869
540:    202A A9 0E                 LDA   #14
540:    202C 8D 00 90              STA   36864
550:    202F A2 10                 LDX   #$10
```

```
560:    2031 8A          CL1       TXA
560:    2032 9D F0 0F              STA   GRALO-$110,X
570:    2035 A9 06                 LDA   #6
570:    2037 9D F0 93              STA   COLLO-16,X
570:    203A E8                    INX
570:    203B D0 F4                 BNE   CL1
580:    203D 60                    RTS
                         ;
610:    203E A9 00       CLEAR     LDA   #0          ; GRAFIK SPEICHER LOESCHEN
620:    2040 A0 11                 LDY   #> GRALO
630:    2042 85 FD                 STA   TMP
640:    2044 84 FE                 STY   TMP+1
645:    2046 A0 00                 LDY   #0
650:    2048 A2 0F                 LDX   #> GRAHI-GRALO   ; ANZAHL PAGES
660:    204A 91 FD       CLR       STA   (TMP),Y
670:    204C C8                    INY
680:    204D D0 FB                 BNE   CLR
690:    204F E6 FE                 INC   TMP+1
700:    2051 CA                    DEX               ; NAECHSTE PAGE
710:    2052 D0 F6                 BNE   CLR
720:    2054 60                    RTS
                         ;
740:    2055 20 FD CE    COLOR     JSR   CHKCOM   ; FARBE SETZEN
750:    2058 20 9E D7              JSR   GETBYT   ; FARBCODE HOLEN
760:    205B A0 00       COL       LDY   #0
770:    205D A9 94                 LDA   #> COLLO
780:    205F 84 FD                 STY   TMP
790:    2061 85 FE                 STA   TMP+1
800:    2063 8A                    TXA               ; FARBCODE
810:    2064 A2 02                 LDX   #> COLHI-COLLO   ; ANZAHL PAGES
820:    2066 91 FD       COL1      STA   (TMP),Y
830:    2068 C8                    INY
840:    2069 D0 FB                 BNE   COL1
850:    206B E6 FE                 INC   TMP+1
860:    206D CA                    DEX               ; NAECHSTE PAGE
870:    206E D0 F6                 BNE   COL1
880:    2070 60                    RTS
                         ;
900:    2071 A0 00       REVERS    LDY   #0          ; GRAFIK INVERTIEREN
910:    2073 A9 11                 LDA   #>GRALO
920:    2075 84 FD                 STY   TMP
930:    2077 85 FE                 STA   TMP+1
940:    2079 A2 0F                 LDX   #> GRAHI-GRALO   ; ANZAHL PAGES
```

```
950:    207B B1 FD    REV1    LDA  (TMP),Y
960:    207D 49 FF            EOR  #$FF      ; ALLE BITS UMDREHEN
970:    207F 91 FD            STA  (TMP),Y
980:    2081 C8               INY
990:    2082 D0 F7            BNE  REV1
1000:   2084 E6 FE            INC  TMP+1
1010:   2086 CA               DEX            ; NAECHSTE PAGE
1020:   2087 D0 F2            BNE  REV1
1030:   2089 60       ILL     RTS            ; AUSSPRUNG BEI UNGUELTIGEN COORDINAT
EN
                      ;
1050:   208A A9 80    RESET   LDA  #$80      ; GRAFIKPUNKT LOESCHEN
1060:   208C 2C               .BYT $2C
1070:   208D A9 00    SET     LDA  #0        ; GRAFIKPUNKT LOESCHEN
1080:   208F 85 97            STA  FLAG
1090:   2091 20 FD CE         JSR  CHKCOM    ; KOMMA
1100:   2094 20 9E D7         JSR  GETBYT    ; X-COORDINATE NACH X-REG
1110:   2097 E0 A8            CPX  #168
1120:   2099 B0 EE            BCS  ILL       ; X-COORDINATE > 167
1130:   209B 86 14            STX  XCOORD
1140:   209D 20 FD CE         JSR  CHKCOM
1150:   20A0 20 9E D7         JSR  GETBYT
1160:   20A3 E0 B0            CPX  #176
1170:   20A5 B0 E2            BCS  ILL
1200:   20A7 8A               TXA            ; Y-COORDINATE IN AKKU
1210:   20A8 4A               LSR
1210:   20A9 4A               LSR
1210:   20AA 4A               LSR
1210:   20AB 4A               LSR
1210:   20AC 0A               ASL            ; DURCH 16 MAL 2
1220:   20AD A8               TAY
                      ; OFFY = 336 * INT(Y/16) + (Y AND 15)
1240:   20AE B9 0E 21         LDA  MULT,Y
1250:   20B1 8D 6B 21         STA  OFFY
1260:   20B4 B9 0F 21         LDA  MULT+1,Y  ; MAL 336
1270:   20B7 8D 6C 21         STA  OFFY+1
1280:   20BA 8A               TXA            ; Y-COORDINATE
1290:   20BB 29 0F            AND  #15
1300:   20BD 18               CLC
1310:   20BE 6D 6B 21         ADC  OFFY
1320:   20C1 8D 6B 21         STA  OFFY
```

```
1320:  20C4 A9 00              LDA  #0
1320:  20C6 8D 6A 21           STA  OFFX+1
                        ; OFFX = 16 * INT(X/8)
1360:  20C9 A5 14              LDA  XCOORD
1360:  20CB 29 F8              AND  #%11111000
1360:  20CD 0A                 ASL
1360:  20CE 8D 69 21           STA  OFFX
1360:  20D1 90 04              BCC  SET0
1360:  20D3 EE 6A 21           INC  OFFX+1
1370:  20D6 18                 CLC            ; AV = GRALO + OFFY +OFFX
1380:  20D7 A9 00      SET0    LDA  #<GRALO
1390:  20D9 6D 6B 21           ADC  OFFY
1400:  20DC 85 FD              STA  AV
1410:  20DE A9 11              LDA  #>GRALO
1420:  20E0 6D 6C 21           ADC  OFFY+1
1430:  20E3 85 FE              STA  AV+1
1440:  20E5 18                 CLC
1450:  20E6 A5 FD              LDA  AV
1460:  20E8 6D 69 21           ADC  OFFX
1470:  20EB 85 FD              STA  AV
1480:  20ED A5 FE              LDA  AV+1
1490:  20EF 6D 6A 21           ADC  OFFX+1
1500:  20F2 85 FE              STA  AV+1
                        ; MA = 2↑((7-X)AND7)
1520:  20F4 A5 14              LDA  XCOORD
1530:  20F6 29 07              AND  #7
1540:  20F8 49 07              EOR  #7
1550:  20FA AA                 TAX
1560:  20FB BD 24 21           LDA  GRBIT,X   ; 2 HOCH MA AUS TABELLE
1570:  20FE A0 00              LDY  #0
1580:  2100 24 97              BIT  FLAG
1590:  2102 10 05              BPL  SET1
1600:  2104 49 FF              EOR  #$FF
1610:  2106 31 FD              AND  (AV),Y   ; PUNKT LOESCHEN
1620:  2108 2C                 .BYT $2C
1630:  2109 11 FD      SET1    ORA  (AV),Y   ; PUNKT SETZEN
1640:  210B 91 FD              STA  (AV),Y
1650:  210D 60                 RTS
                        ;
                        ; MULTIPLIKATIONSTABELLE N*336, N=0,10
1675:  210E            MULT    =    *
1680:  210E            COUNT   ←    0
1690:  210E 00 00              .WOR COUNT*336
1700:  2110            COUNT   ←    COUNT+1
1710:  2110                    .IF  COUNT-11
```

```
1710:  2110                     .GOTO1690
1690:  2110 50 01               .WOR COUNT*336
1700:  2112           COUNT     ←     COUNT+1
1710:  2112                     .IF  COUNT-11
1710:  2112                     .GOTO1690
1690:  2112 A0 02               .WOR COUNT*336
1700:  2114           COUNT     ←     COUNT+1
1710:  2114                     .IF  COUNT-11
1710:  2114                     .GOTO1690
1690:  2114 F0 03               .WOR COUNT*336
1700:  2116           COUNT     ←     COUNT+1
1710:  2116                     .IF  COUNT-11
1710:  2116                     .GOTO1690
1690:  2116 40 05               .WOR COUNT*336
1700:  2118           COUNT     ←     COUNT+1
1710:  2118                     .IF  COUNT-11
1710:  2118                     .GOTO1690
1690:  2118 90 06               .WOR COUNT*336
1700:  211A           COUNT     ←     COUNT+1
1710:  211A                     .IF  COUNT-11
1710:  211A                     .GOTO1690
1690:  211A E0 07               .WOR COUNT*336
1700:  211C           COUNT     ←     COUNT+1
1710:  211C                     .IF  COUNT-11
1710:  211C                     .GOTO1690
1690:  211C 30 09               .WOR COUNT*336
1700:  211E           COUNT     ←     COUNT+1
1710:  211E                     .IF  COUNT-11
1710:  211E                     .GOTO1690
1690:  211E 80 0A               .WOR COUNT*336
1700:  2120           COUNT     ←     COUNT+1
1710:  2120                     .IF  COUNT-11
1710:  2120                     .GOTO1690
1690:  2120 D0 0B               .WOR COUNT*336
1700:  2122           COUNT     ←     COUNT+1
1710:  2122                     .IF  COUNT-11
1710:  2122                     .GOTO1690
1690:  2122 20 0D               .WOR COUNT*336
1700:  2124           COUNT     ←     COUNT+1
1710:  2124                     .IF  COUNT-11
1720:  2124 01 02 04 GRBIT      .BYT 1,2,4,8,$10,$20,$40,$80   ; ZWEIERPOTENZEN
```

```
                             ;
1740:    212C 20 FD CE GSAVE      JSR   CHKCOM   ; GRAFIK ABSPEICHERN
1750:    212F 20 D1 E1            JSR   GETPAR   ; FILENAMEN UND GERAETEADRESSE HOLEN
1760:    2132 A2 00               LDX   #< GRAHI
1770:    2134 A0 20               LDY   #> GRAHI
1780:    2136 A9 00               LDA   #< GRALO
1790:    2138 85 FD               STA   TMP
1800:    213A A9 11               LDA   #> GRALO
1810:    213C 85 FE               STA   TMP+1
1820:    213E A9 FD               LDA   #TMP
1830:    2140 85 B9               STA   SA       ; ABSOLUT SPEICHERN
1840:    2142 4C D8 FF            JMP   SAVE
                             ;
1860:    2145 20 FD CE GLOAD      JSR   CHKCOM   ; GRAFIK LADEN
1865:    2148 20 D1 E1            JSR   GETPAR   ; FILENAMEN UND GERAETEADRESSE HOLEN
1870:    214B A9 01               LDA   #1       ; ABSOLUT LADEN
1880:    214D 85 B9               STA   SA
1890:    214F A9 00               LDA   #0       ; LOAD FLAG
1900:    2151 4C D5 FF            JMP   LOAD
                             ;
1920:    2154 A9 0C        GOFF   LDA   #12
1920:    2156 8D 00 90            STA   36864
1930:    2159 A9 16               LDA   #22
1930:    215B 8D 02 90            STA   36866
1940:    215E A9 AE               LDA   #174
1940:    2160 8D 03 90            STA   36867
1950:    2163 A9 C0               LDA   #192
1950:    2165 8D 05 90            STA   36869
1960:    2168 60                  RTS
                             ;
2000:    216B              OFFX   *=    *+2
2010:    216D              OFFY   *=    *+2
```

```
100 FORI=8192TO8552:READA:POKEI,A:B=B+A
110 NEXTI
120 IFB<>43663THENPRINT"FEHLER!!!"
1000 DATA  76 , 27 , 32 , 76 , 62 , 32 , 76 , 85
1010 DATA  32 , 76 , 113 , 32 , 76 , 141 , 32 , 76
1020 DATA  138 , 32 , 76 , 69 , 33 , 76 , 44 , 33
1030 DATA  76 , 84 , 33 , 169 , 151 , 141 , 3 , 144
1040 DATA  169 , 21 , 141 , 2 , 144 , 169 , 204 , 141
1050 DATA  5 , 144 , 169 , 14 , 141 , 0 , 144 , 162
1060 DATA  16 , 138 , 157 , 240 , 15 , 169,6,157
1070 DATA  240 , 147, 232,208,244,96,169,0
1080 DATA  160 , 17 , 133 , 253 , 132 , 254 , 160 , 0
1090 DATA  162 , 15 , 145 , 253 , 200 , 208 , 251 , 230
1100 DATA  254 , 202,208,246,96,32,253,206
1110 DATA  32 , 158 , 215 , 160 , 0 , 169 , 148 , 132
1120 DATA  253 , 133,254,138,162,2,145,253
1130 DATA  200 , 208,251,230,254,202,208,246
1140 DATA  96 , 160 , 0 , 169 , 17 , 132 , 253 , 133
1150 DATA  254 , 162,15,177,253,73,255,145
1160 DATA  253 , 200,208,247,230,254,202,208
1170 DATA  242 , 96 , 169 , 128 , 44 , 169,0,133
1180 DATA  151 , 32 , 253 , 206 , 32 , 158,215,224
1190 DATA  168 , 176,238,134,20,32,253,206
1200 DATA  32 , 158 , 215 , 224 , 176 , 176 , 226 , 138
1210 DATA  74 , 74 , 74 , 74 , 10 , 168 , 185 , 14
1220 DATA  33 , 141 , 107 , 33 , 185 , 15 , 33 , 141
1230 DATA  108 , 33 , 138 , 41 , 15 , 24 , 109 , 107
1240 DATA  33 , 141 , 107 , 33 , 169 , 0 , 141 , 106
1250 DATA  33 , 165 , 20 , 41 , 248 , 10 , 141 , 105
1260 DATA  33 , 144 , 4 , 238 , 106 , 33 , 24 , 169
1270 DATA  0 , 109 , 107 , 33 , 133 , 253 , 169 , 17
1280 DATA  109 , 108,33,133,254,24,165,253
1290 DATA  109 , 105,33,133,253,165,254,109
1300 DATA  106 , 33 , 133 , 254 , 165 , 20,41,7
```

```
1310 DATA  73 , 7 , 170 , 189 , 36 , 33 , 160 , 0
1320 DATA  36 , 151 , 16 , 5 , 73 , 255 , 49 , 253
1330 DATA  44 , 17 , 253 , 145 , 253 , 96 , 0 , 0
1340 DATA  80 , 1 , 160 , 2 , 240 , 3 , 64,5
1350 DATA  144 , 6 , 224 , 7 , 48 , 9 , 128 , 10
1360 DATA  208 , 11 , 32 , 13 , 1 , 2 , 4 , 8
1370 DATA  16 , 32 , 64 , 128 , 32 , 253 , 206 , 32
1380 DATA  209 , 225,162,0,160,32,169,0
1390 DATA  133 , 253,169,17,133,254,169,253
1400 DATA  133 , 185,76,216,255,32,253,206
1410 DATA  32 , 209 , 225 , 169 , 1 , 133 , 185 , 169
1420 DATA  0 , 76 , 213 , 255 , 169 , 12 , 141 , 0
1430 DATA  144 , 169,22,141,2,144,169,174
1440 DATA  141 , 3 , 144 , 169 , 192 , 141,5,144
1450 DATA  96 , 255 , 14 , 255 , 0 , 255 , 5 , 255
```

Anwendung der Grafikhilfe :

Am Anfang des Grafikprogramms steht eine Sprungtabelle für die verschiedenen Funktionen. Am einfachsten können Sie alle Funktionen anwenden, wenn Sie sich zu Beginn Ihres Programms ebenfalls eine solche Tabelle anlegen. Dies könnte so aussehen:

```
10 IN=2*4096:REM SYS IN - GRAFIK EINSCHALTEN
20 CL=IN+3   :REM SYS CL - GRAFIK LöSCHEN
30 CO=IN+6   :REM SYS CO,PF -  PUNKTFARBE SETZEN
40 RV=IN+9   :REM SYS RV - GRAFIK INVERTIEREN
50 SE=IN+12 :REM SYS SE,X,Y - GRAFIKPUNKT SETZEN
60 RS=IN+15 :REM SYS RS,X,Y - GRAFIKPUNKT LöSCHEN
70 GL=IN+18 :REM SYS GL,"NAME",1 ODER 8 - GRAFIK LADEN
80 GS=IN+21 :REM SYS GS,"NAME",1 ODER 8 - GRAFIK ABSPEICHERN
90 OF=IN+24 :REM SYS OF - GRAFIKMODUS AUSSCHALTEN
```

Dank der Grafikhilfe haben Sie nun eine echte Punktgrafik zur Verfügung. Sie können in der waagerechten (X-) Richtung 168 und in der senkrechten (Y-) Richtung 176 einzelne Punkte ansteuern. Ein Punkt wird gesetzt, in dem Sie zum Beispiel SYS SE,84,88 eingeben. Die würde einen Punkt in der Mitte des Bildschirms setzen. Mit SYS IN schalten Sie den Grafikmodus ein, mit SYS CL löschen Sie die Grafikseite. Mit SYS CO,PF bestimmen Sie die Farbe aller Punkte. PF darf Werte zwischen 0 und 7 haben. SYS RV invertiert die gesamte Grafik. Mit SYS RS,84,88 können Sie den Punkt in der Mitte des Bildschirms wieder löschen. Mit SYS OF schalten Sie wieder den Textmodus ein. Sie können die gesamte Grafikseite auf Kassette oder Diskette abspeichern und wieder laden. Dies geschieht mit den beiden Befehlen SYS GS,"NAME",1 oder 8 zum Abspeichern und SYS GL,"NAME",1 oder 8 zum Laden. Damit haben Sie die wichtigsten Funktionen, Setzen und Löschen

eines Punktes, Wählen der Farbe, Grafik ein- und ausschalten und Bilder laden bzw. abspeichern realisiert. Sicher gibt es noch viele Möglichkeiten, diese Grafikhilfe zu erweitern. Dies Programm ist als Anregung für Sie gedacht. Erweiterungen steht nur Ihre Fantasie entgegen.
Unser Sinuskurveprogramm von Seite sieht nun so aus, wie Sie es unten sehen. Sie werden feststellen, daß es um einiges schneller und kürzer geworden ist. Tippen Sie das Programm ein, nachdem Sie die Grafikhilfe geladen haben.

```
10 IN=2*4096
20 CL=IN+3
30 CO=IN+6
40 SE=IN+12
50 OF=IN+24
60 SYS IN:SYS CL:SYS CO,6
100 FORI=-PITO2*PISTEP2*PI/168:REM BENUTZEN SIE DAS PI-ZEICHEN AUF
    IHRER TASTATUR
110 X=U:Y=SIN(I)*88+88
120 SYS SE,U,Y
130 U=U+1
140 NEXTI
150 GETA$:IFA$=""THEN150
160 SYS OF:END
```

3.6 Funktionenplotter mit Luxus

Auf der folgenden Seite finden Sie nun einen weiteren Funktionenplotter. Da wir etwas mehr Platz zur Verfügung haben als in der Grundversion, können wir diesen auch etwas komfortabler gestalten. So müssen Sie die Funktionen nicht mehr in die Zeilen hineinschreiben, sondern können Sie per INPUT eingeben. Das Programm modifiziert sich dann selber, indem es die Funktion in Zeile 20 schreibt. Da das Programm sich selber ändert, müssen Sie zwei Dinge beim Abtippen beachten: Ändern Sie auf keine Fälle Zeile 10 und 20 oder fügen Sie irgendetwas vor Zeile 20 ein! Wenn Sie das tun, stimmen die Adressen für die POKE-Anweisungen nicht mehr und schlimmstenfalls könnte sich das Programm selbst zerstören. Speichern Sie zweitens das Programm unbedingt zuerst ab, bevor Sie es laufen lassen. Ein Tippfehler kann in diesem Programm sehr fatale Folgen haben.
Zur Benutzung des Programms:
Der Funktionenplotter arbeitet interaktiv, d.h. er fragt Sie klar und deutlich nach allen Eingaben, so daß Sie praktisch nichts falsch machen können. Als erstes geben Sie die Funktion ein, die gezeichnet werden soll, z.B. SIN(X) . Als Variable müssen Sie immer "X" verwenden. Anschließend wird diese Funktion in Zeile 20 geschrieben. Dies geschieht in den Zeilen 70 - 140. Sie geben nun Anfangs- und Endpunkt der Funktion ein. Weiter können Sie wählen, ob Sie die Maximalwerte für Y selber bestimmen wollen. Nach allen Eingaben dauert es einen Moment, bis der Graph erscheint, da das Programm die gesamte Funktion erst einmal durchrechnet. Wenn der Graph fertig ist, können Sie durch Drücken einer Taste die Arbeit fortsetzen, entweder mit der gleichen Funktion unter neuen Bedingungen oder mit einer neuen Funktion.

```
10 GOTO21
20 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
21 IN=2*4096
22 CL=IN+3
23 CO=IN+6
24 RV=IN+9
25 SE=IN+12
26 RE=IN+15
27 OF=IN+24
28 SYS CO,3*16+3
30 PRINTCHR$(147):PRINT" * FUNKTIONENPLOTTER *":PRINT:PRINT
40 DIMY(167):YM=1E37
50 PRINT"GEBEN SIE NUN DIE FUNKTION EIN F(X)= ":INPUTF$
60 F$="Y="+F$:M=8717
70 FORI=1TOLEN(F$):A$=MID$(F$,I,1)
80 RESTORE:FORJ=1TO6:READB$,B:IFA$=B$THENF=B:GOTO130
90 NEXTJ
100 A$=MID$(F$,I,3):I=I+2
110 FORJ=1TO10:READB$,B:IFA$=B$THENF=B:GOTO130
115 NEXTJ
120 I=I-2:F=ASC(MID$(F$,I,1))
130 POKEM,F:M=M+1:NEXTI
140 POKEM,58:POKEM+1,142:POKEM+2,58:POKEM+3,143
200 PRINT:PRINT"BITTE GEBEN SIE NUN XL EIN : ":INPUTXL
210 PRINT:PRINT"BITTE GEBEN SIE NUN XR EIN : ":INPUTXR
220 IFXL>XRTHEN200
230 PRINT:PRINT"WOLLEN SIE DIE Y-WERTE BESTIMMEN (J/N) ?"
240 GETY$:IFY$=""THEN240
250 IFY$="N"THEN290
260 IFY$<>"J"THEN240
270 PRINT:PRINT"GEBEN SIE BITTE YU EIN : ":INPUTYU
280 PRINT:PRINT"GEBEN SIE BITTE YO EIN : ":INPUTYO
290 IFYU>YOTHEN270
340 PRINT"BITTE WARTEN SIE EINEN MOMENT"
350 PRINT:PRINT"WENN DER GRAPH FERTIG IST, KOENNEN SIE DURCH DRUECKEN "
360 PRINT"EINER TASTE FORTFAHREN"
400 IFY$="J"THEN475
410 DX=(XR-XL)/167:Y1=YM:Y2=-YM
```

```
430 X=XL:FORI=0TO167
431 IFX=0THEN440
432 GOSUB20:Y(I)=Y
440 X=X+DX:IFY=YMTHEN470
450 IFY<Y1THENY1=Y
460 IFY>Y2THENY2=Y
470 NEXTI
475 SYS IN :SYS CL
480 IFY$="N"THENYU=Y1:YO=Y2
500 DX=(XR-XL)/167:DY=(YO-YU)/176:M=(XR-XL)/DX
510 B=176.5+YU/DY:A=.5+M
520 IFB>=0ANDB<=177THENFORI=0TOA:SYSSE,I,B:NEXT:GOTO540
530 B=180
540 A=.5-XL/DX:B=176.5-(YO-YU)/DY
550 IFA>=0ANDA<=169THENFORI=BTO176:SYS SE,A,I:NEXT
600 X=XL:Z=176.5
610 FORI=0TOM
630 Y=Y(I)
640 B=Z-(Y-YU)/DY
650 IFB>=0ANDB<=177THENSYSSE,I,B
660 X=X+DX:NEXTI
700 GETT$:IFT$=""THEN700
705 SYS OF
720 PRINTCHR$(147):PRINT:PRINT"NOCH EINMAL UNTER ANDEREN BEDINGUNGEN (J/N) "
730 GETN$:IFN$=""THEN730
740 IFN$="N"THEN760
750 GOTO200
760 PRINT:PRINT"NOCH EINE FUNKTION (J/N)"
770 GET N$:IFN$=""THEN770
780 IFN$="N"THENEND
820 GOTO50
2000 DATA +,170,-,171,*,172,/,173,↑,174,=,178
2010 DATA SGN,180,INT,181,ABS,182,SQR,186,RND,187,LOG,188,EXP,189
2020 DATA COS,190,SIN,191,TAN,192,ATN,193

READY.
```

3.7 Grafikeditor

Das folgende Programm ermöglicht es Ihnen mit Hilfe der Cursortasten oder des Joysticks auf dem Bildschirm zu zeichnen. Weiterhin können Sie mit Hilfe der Funktionstasten die Farben ändern, Bilder abspeichern und laden. Den Grafikeditor können Sie benutzen, um Bilder zu malen oder um Grafiken für Spiele zu gestalten.

Die Tasten sind wie folgt belegt:

Cursortasten und Joystick : Bewegung des Cursors in alle 4 Richtungen

Taste F1 : Farbe der Punkte ändern

Taste F3 : Farbe des Rahmens ändern

Taste F5 : Farbe des Hintergrunds ändern

Taste F7 : Schalter für Modus Zeichnen/Löschen

Taste F2 : Laden eines Bildes

Taste F4 : Speichern eines Bildes

Taste F8 : Bild löschen

Zu Beginn des Programms müssen Sie angeben, wie Ihr Bild heißen soll und ob Sie mit Datasette oder Diskettenlaufwerk arbeiten. Dann können Sie zeichnen oder ein Bild unter dem angegebenen Namen laden. Mit der Taste F7 wählen Sie, ob Sie den Cursor nur bewegen oder ob Sie auch zeichnen wollen. Sie können jederzeit die Farben auf dem Bildschirm ändern, indem Sie einfach die entsprechenden Funktionstasten drücken.

Das Programm ist ganz einfach aufgebaut und erlaubt Ihnen daher ohne Schwierigkeiten eigene Erweiterungen einzubauen.

```
10 GOSUB1000:X=0:Y=0
16 IN=2*4096
17 CL=IN+3
18 CO=IN+6
19 RV=IN+9
20 SE=IN+12
21 RE=IN+15
25 GL=IN+18
26 GS=IN+21
27 OF=IN+24
28 SYS IN:SYS CL:SYS CO,3
30 GETA$:SYSSE,X,Y
35 IFS=0THENSYS RE,X,Y
40 IFA$=CHR$(17)THENY=Y+1:SYSSE,X,Y:GOTO100
50 IFA$=CHR$(145)THENY=Y-1:SYS SE,X,Y:GOTO100
60 IFA$=CHR$(29)THENX=X+1:SYS SE,X,Y:GOTO100
70 IFA$=CHR$(157)THENX=X-1:SYS SE,X,Y:GOTO100
80 IFA$=CHR$(133)THENF=(F+1)AND7 :SYS CO,F
90 IFA$=CHR$(136)THENS=1+(S=1)
100 IFA$=CHR$(134)THENH1=(H1+1)AND7:H=H1+H3:POKE36879,H
120 IFA$=CHR$(135)THENH2=(H2+1)AND15:H3=H2*16+8:H=H1+H3:POKE36879,H
130 IFA$=CHR$(137)THENSYS GL,N$,D
140 IFA$=CHR$(138)THEN SYS GS,N$,D
150 IFA$=CHR$(140)THEN SYS CL
160 POKE37154,127:P2=PEEK(37152)AND128
170 P1=PEEK(37151):POKE37154,255
180 IFP2=0THENX=X+1:SYSSE,X,Y:GOTO999
190 IF(P1AND16)=0THENX=X-1:SYSSE,X,Y:GOTO999
200 IF(P1AND8)=0THENY=Y+1:SYSSE,X,Y:GOTO999
210 IF(P1AND4)=0THENY=Y-1:SYSSE,X,Y:GOTO999
999 GOTO30
1000 PRINT"GEBEN SIE DEN NAMEN DES BILDES EIN":INPUTN$
1020 PRINT"ARBEITEN SIE MIT DISKETTE ODER KASSETTE (D/K)":INPUTD$
1040 IFD$="D"THEND=8:RETURN
1050 IFD$="K"THEND=1:RETURN
1060 GOTO1020
```

3.8 Grafiken zum Anschauen

Zum guten Schluß möchten wir Ihnen noch ein paar kleine Programme zeigen, die Grafiken auf dem Bildschirm erzeugen, die Sie sicher kennen, von denen Sie aber bis jetzt nicht wußten, wie diese gemacht werden. Wir verzichten hier bewußt auf mathematische Erklärungen, wie und warum die Grafiken entstehen. Diese kleinen Programme sollen Sie lediglich erfreuen und anregen, sich doch mit dem faszinierenden Gebiet der Computergraphik zu beschäftigen.

Alle Programme fragen zu Beginn nach einem Namen für das Bild und ob Sie mit Diskette oder Kassette arbeiten. Nach Fertigstellung können Sie das Bild dann abspeichern. Sie brauchen dafür nur "S" einzutippen. Durch Betätigen der Taste "E" können Sie das Programm beenden ohne das Bild abzuspeichern.

Voraussetzung für all diese Programme ist, daß Sie die Grafikhilfe geladen haben.

Das erste Programm erzeugt ein Linienmuster auf dem Bildschirm:

```
10 IN=2*4096
11 CL=IN+3
12 CO=IN+6
13 SE=IN+12
14 GS=IN+21
15 OF=IN+24
25 GOSUB 600
28 POKE36879,8
29 SYS IN:SYS CL: SYS CO,1
30 FORX1=0TO168STEP20
40 FORX2=0TO168STEP20
50 DX=(X2-X1)/178:X=X1
```

```
60 FORY=0TO176
70 X=X+DX
80 SYS SE,X,Y
90 NEXTY,X2,X1
500 GETA$:IFA$=""THEN500
520 IFA$="E"THENEND
530 IFA$<>"S"THEN500
540 SYS OF:POKE36879,27
550 IFD$="K"THENSYS GS,N$,1
560 IFD$="D"THENSYS GS,N$,8
570 END
600 INPUT"NAME";N$
610 INPUT"KASETTE ODER DISKETTE (K/D) ";D$
620 RETURN
```

Wenn Sie die Schrittweite in den Zeilen 30 und 40 variieren, erhalten Sie weitere schöne Muster. Nehmen Sie in den beiden Zeilen auch einmal verschiedene Werte.

Der Effekt des nächsten Programms hängt hauptsächlich vom Wert der Variablen K ab. Wählen Sie für K einmal sehr kleine Werte von 1 bis 10, zum andern sehr große wie z.B. 60 oder 270.

```
21 IN=2*4096
22 CL=IN+3
23 CO=IN+6
24 SE=IN+12
25 GS=IN+21
26 OF=IN+24
27 PI=3.14159265
28 GOSUB600
29 POKE36879,8
30 INPUT"KONSTANTE";K:V=10
```

```
35 SYS IN:SYS CL: SYS CO,1
40 FORI=0TO2*PISTEP2*PI/672
50 R=V*SIN(K*I)
60 X=R*7*COS(K)+83
70 Y=R*7*SIN(I)+87
80 SYS SE,X,Y
90 NEXTI
500 GETA$:IFA$=""THEN100
520 IFA$="E"THENEND
530 IFA$<>"S"THEN500
540 SYS OF:POKE36879,27
550 IFD$="K"THENSYS GS,N$,1
560 IFD$="D"THENSYS GS,N$,8
570 END
600 INPUT"NAME";N$
610 INPUT"KASETTE ODER DISKETTE (K/D) ";D$
620 RETURN
```

V ist hier ein Vergrößerungsfaktor, die Schrittweite in Zeile 40 bestimmt, wie dicht die einzelnen Punkte aneinanderliegen. Beide Werte können Sie nach eigenem Geschmack verändern.

Das nächste Programm erzeugt eine Spirale. Der Effekt hängt hier hauptsächlich von der Variablen E ab. Sehr kleine Werte für E wie z.B. 0.05 ergeben eine sehr schöne Spirale. Ein völlig anderer Effekt ergibt sich, wenn Sie für E Werte zwischen 1 und 5 wählen. Doch lassen Sie sich überraschen:

X

```
21 IN=2*4096
22 CL=IN+3
23 CO=IN+6
24 SE=IN+12
25 GS=IN+21
26 OF=IN+24
28 GOSUB 600
29 POKE36879,8
30 INPUT"KONSTANTE ";E
35 SYS IN:SYS CL: SYS CO,1
40 FORI=0TO200STEPE
50 X=I*COS(I)+84
60 Y=I*SIN(I)+88
70 IFX<0ORX>167THEN95
80 IFY<0ORY>175THEN95
90 SYS SE,X,Y
95 NEXTI
500 GETA$:IFA$=""THEN100
520 IFA$="E"THENEND
530 IFA$<>"S"THEN500
540 SYS OF:POKE36879,27
550 IFD$="K"THENSYS GS,N$,1
560 IFD$="D"THENSYS GS,N$,8
570 END
600 INPUT"NAME";N$
610 INPUT"KASETTE ODER DISKETTE (K/D) ";D$
620 RETURN
```

Das nachfolgende Programm erzeugt in sich verdrehte Rechtecke.

```
20 REM *** RECHTECKE ***
21 IN=2*4096
22 CL=IN+3
23 CO=IN+6
24 SE=IN+12
25 GS=IN+21
26 OF=IN+24
28 GOSUB600
29 POKE36879,8
30 SYS IN:SYS CL:SYS CO,1
31 X(1)=0:Y(1)=0:X(2)=167:Y(2)=0
35 X(3)=167:Y(3)=175:X(4)=0:Y(4)=175
40 X(5)=X(1):Y(5)=Y(1):H=.5:K=15
50 FORI=1TO40:FORJ=1TO4
100 X1=X(J):Y1=Y(J)
130 X2=X(J+1):Y2=Y(J+1)
135 IFY1=Y2THENFORX=X1TOX2STEP1+2*(X1>X2):SYSSE,X,Y1:NEXTX:GOTO190
140 DX=(X2-X1)/ABS(Y1-Y2):X=X1
150 FORY=Y1TOY2STEP1+2*(Y1>Y2)
160 SYS SE,X,Y:X=X+DX
180 NEXTY
190 NEXTJ
200 FORJ=1TO4
210 XX(J)=X(J)+INT((X(J+1)-X(J))/K+H)
220 YY(J)=Y(J)+INT((Y(J+1)-Y(J))/K+H):NEXTJ
250 FORJ=1TO4:X(J)=XX(J):Y(J)=YY(J):NEXTJ
260 X(5)=X(1):Y(5)=Y(1)
270 NEXTI
500 GET A$:IFA$=""THEN500
520 IFA$="E"THENEND
530 IFA$<>"S"THEN500
540 SYS OF:POKE36879,27
550 IFD$="K"THENSYS GS,N$,1
560 IFD$="D"THENSYS GS,N$,8
570 END
600 INPUT"NAME";N$
610 PRINT"KASSETTE ODER DISKETTE (K/D)":INPUTD$
620 RETURN
```

Gehen wir einen Schritt weiter. Das Programm auf der nächsten Seite erzeugt in sich verdrehte Sechsecke. Die Schleife in Zeile 100 bestimmt die Anzahl der Sechsecke, die auf dem Bildschirm gezeichnet werden. Durch Verändern der Variablen W in Zeile 40 können Sie auch Achtecke oder Figuren mit noch mehr Ecken erzeugen. Sie müssen dann aber auch die Schleifen in den Zeilen 50,110,200 und 250 verändern. Ferner müssen Sie eventuell Felder für X, MX, Y und MY dimensionieren und die Zeile 240 modifizieren.

Ein weiteres sehr schönes Programm finden Sie auf der übernächsten Seite. Es erzeugt eine ellipsenförmige Figur mit 20 Eckpunkten, in der jeder Eckpunkt mit jedem Eckpunkt durch eine Strecke verbunden ist.

Mit den beiden letzten Programmen wagen wir uns in den Bereich der dreidimensionalen Grafik. Die beiden Programme sind identisch aufgebaut, lediglich die Definition der Funktionen in Zeile 30 ist verschieden. Die beiden Programme laufen relativ lange. Haben Sie also bitte etwas Geduld. Bedenken Sie, daß mehrere Tausend Punkte berechnet werden müssen.

```
20 REM SECHSECKE
21  IN=2*4096
22 CL=IN+3
23 CO=IN+6
24 SE=IN+12
25 GS=IN+21
26 OF=IN+24
28 SYS  IN:SYS CL:SYS CO,8*16+3
30 DIMX(6),Y(6),MX(6),MY(6)
40 R=84:U=84:V=88:H=.5:W=60*π/180
50 FORJ=0TO6:W1=J*W
60 X(J)=INT(U-R*COS(W1)+H)
70 Y(J)=INT(V+R*SIN(W1)+H)
80 NEXTJ
100 FORN=1TO20
110 FORJ=0TO5
130 X1=X(J):Y1=Y(J):X2=X(J+1):Y2=Y(J+1)
135 IFY1=Y2THENFORX=X1TOX2STEP1+2*(X1>X2):SYSSE,X,Y1,NEXTX:GOTO190
140 DX=(X2-X1)/ABS(Y1-Y2):X=X1
150 FORY=Y1TOY2STEP1+2*(Y1>Y2)
160 SYS SE,X,Y
170 X=X+DX
180 NEXTY
190 NEXTJ
200 FORI=0TO5
210 MX(I)=INT((X(I)+X(I+1))/2+H)
220 MY(I)=INT((Y(I)+Y(I+1))/2+H)
230 NEXTI
240 MX(6)=MX(0):MY(6)=MY(0)
250 FORI=0TO6
260 X(I)=MX(I):Y(I)=MY(I)
270 NEXTI
280 NEXTN
```

```
20 REM ELLIPSE
21 IN=2*4096
22 CL=IN+3
23 CO=IN+6
24 SE=IN+12
25 GS=IN+21
26 OF=IN+24
28 GOSUB600
29 POKE36879,8
30 SYS IN:SYS CL:SYS CO,1
35 A=83:B=87:N=20:DIMX(20),Y(20)
40 R=83:U=84:V=87:H=.5:W=360/N*π/180
50 FORJ=0TON-1:W1=J*W
60 X(J)=INT(U+A*COS(W1)+H)
70 Y(J)=INT(V-B*SIN(W1)+H)
80 NEXTJ
100 FORI=0TON-2:X1=X(I) :Y1=Y(I)
110 FORJ=I+1TON-1
130 X2=X(J):Y2=Y(J)
135 IFY1=Y2THENFORX=X1TOX2STEP1+2*(X1>X2):SYSSE,X,Y1:NEXTX:GOTO190
140 DX=(X2-X1)/ABS(Y1-Y2):X=X1
150 FORY=Y1TOY2STEP1+2*(Y1>Y2)
160 SYS SE,X,Y
170 X=X+DX
180 NEXTY
190 NEXTJ
270 NEXTI
500 GET A$:IFA$=""THEN500
520 IFA$="E"THENEND
530 IFA$<>"S"THEN500
540 SYS OF:POKE36879,27
550 IFD$="K"THENSYS GS,N$,1
560 IFD$="D"THENSYS GS,N$,8
570 END
600 INPUT"NAME";N$
610 PRINT"KASSETTE ODER DISKETTE (K/D)":INPUTD$
620 RETURN
```

```
21 IN=2*4096
22 CL=IN+3
23 CO=IN+6
24 RV=IN+9
25 SE=IN+12
26 GS=IN+21
27 OF=IN+24
28 GOSUB600
29 POKE36879,8
30 SYS IN:SYS CL:SYS CO,1
35 DEFFNA(Z)=90*EXP(-Z*Z/1500)
40 FORA=1TO2
50 G=84:K=50:FORX=-83TO0:L=-50:H=5*INT(SQR(10000-X*X)/5)
60 FORY=HTO-HSTEP-5:Z=25+FNA(SQR(X*X+Y*Y))-.6*Y
65 X1=G+X:Y1=176-Z-K:X2=G-X
70 IFZ>LTHENL=Z:SYS SE,X1,Y1:SYS SE,X2,Y1
80 NEXTY,X
500 GET A$:IFA$=""THEN500
510 SYS OF:POKE36879,27
520 IFA$="E"THENEND
530 IFA$<>"S"THEN500
540 SYS OF:POKE36879,27
550 IFD$="K"THENSYS GS,N$,1
560 IFD$="D"THENSYS GS,N$,8
570 END
600 INPUT"NAME";N$
610 PRINT"KASSETTE ODER DISKETTE (K/D)":INPUTD$
620 RETURN

READY.
```

```
21 IN=2*4096
22 CL=IN+3
23 CO=IN+6
24 RV=IN+9
25 SE=IN+12
26 GS=IN+21
27 OF=IN+24
28 GOSUB600
29 POKE36879,8
30 SYS IN:SYS CL:SYS CO,1
35 DEFFNA(Z)=38*(SIN(Z/24)+.48*SIN(3*Z/24))+20
40 FORA=1TO2
50 G=84:K=50:FORX=-83TO0:L=-50:H=5*INT(SQR(10000-X*X)/5)
60 FORY=HTO-HSTEP-5:Z=25+FNA(SQR(X*X+Y*Y))-.6*Y
65 X1=G+X:Y1=176-Z-K:X2=G-X
70 IFZ>LTHENL=Z:SYS SE,X1,Y1:SYS SE,X2,Y1
80 NEXTY,X
500 GET A$:IFA$=""THEN500
510 SYS OF:POKE36879,27
520 IFA$="E"THENEND
530 IFA$<>"S"THEN500
540 SYS OF:POKE36879,27
550 IFD$="K"THENSYS GS,N$,1
560 IFD$="D"THENSYS GS,N$,8
570 END
600 INPUT"NAME";N$
610 PRINT"KASSETTE ODER DISKETTE (K/D)":INPUTD$
620 RETURN

READY.
```

3.9 Die Supererweiterung des VC-20

Wie Sie sicher wissen, besteht die Supererweiterung VC-1211A aus zwei Teilen

- einer 3K Speichererweiterung
- dem Superexpander als 2K Befehlserweiterung

Die Supererweiterung des VC-20 erweitert sowohl die graphischen, als auch die musikalischen Möglichkeiten des VC-20 um ein Vielfaches. Sie kann mit normalen Basic Befehlen programmiert werden. Die Programme, die auf einer Supererweiterung geschrieben wurden, laufen auch nur mit selbiger. Dies gilt auch, wenn Sie eine Supererweiterung zusammen mit einem Speichererweiterungsmodul eingesteckt haben. Die Erweiterung stellt Ihnen über 15 neue Befehle zur Verfügung, die es Ihnen erlauben, sehr einfach und wirkungsvoll zu arbeiten.

Die Supererweiterung teilt den VC-20 Bildschirm in 1024 mal 1024 Punkte ein. Wenn Sie nun den folgenden Befehl eingeben, wird auf der Mitte des Bildschirmes ein Punkt erscheinen: GRAPHIC2 : POINT 1,512,512 Es muß jedoch beachtet werden, daß es sich hier um Pseudo-Koordinaten handelt. Sie geben nicht die tatsächliche Auflösung der Erweiterung an, sondern einen internen Standard,der in die wirkliche Auflösung von 160 mal 160 Punkten umgerechnet wird.

Sicher haben auch Sie schon das Problem gehabt, daß Sie ein Programm hatten, das nur für eine normale 3k Ram Erweiterung geschrieben wurde und nun nicht mit der Supererweiterung läuft. Es besteht aber die Möglichkeit, den Superexpander abzuschalten. Dies geschieht durch die einfache Eingabe eines SYS Befehls. Dieser lautet: SYS 64818 (SYS steht immer für System). Nach Eingabe dieses Befehls können Sie nun Ihre 3k Programme wie gewohnt einladen.

Eingeschaltet wird der Superexpander wiederum mit einem SYS Befehl: SYS 64802. Dieser SYS Befehl ist ein System Reset

Befehl,das heißt, der VC-20 befindet sich im selben Zustand, wie nach dem Einschalten. Sie können dieses kontrollieren, indem Sie die Funktionstasten betätigen. Es werden nun wieder wie gewohnt die Expanderbefehle erscheinen. Auf den nun folgenden Seiten werden wir Ihnen einige Beispiele zeigen, die es Ihnen ermöglichen, wirklich alles aus dieser, wie wir meinen, einzigartigen Erweiterung herauszuholen. Viel Spaß beim Eingeben und Ausprobieren. Es folgt nun ein kleines Programm,das Ihnen die Vielzahl der Farbmöglichkeiten darlegt.

```
100 PRINT "(CLR/HOME) "
110 DEF FNC(V) =V*16-8
120 H=36879
130 FOR BK= 1 TO 16
140 PRINT "( CLEAR ) ( WHT) "
150 IF BK > 1 THEN PRINT " ( BLK) " ;
160 PRINT " HINTERGRUND"
165 PRINT " (CRSR DOWN ) " BK
170 FOR BD = 0 TO 7
180 POKE H, FNC (BK) + BD
190 PRINT , " RAHMEN " ; BD
200 FOR W = 1 TO 500 : NEXT W
210 NEXT BD
220 NEXT BK
230 POKE H, 27
240 END
```

Nun haben Sie die vielen Möglichkeiten der Supererweiterung gesehen. Es folgen nun noch Programme, die Ihnen als kleiner Denkanstoß für Ihre eigenen Programme dienen sollen. Als erstes ein Programm das eine Sinus-Kurve mit dem dazugehörigen Koordinatenkreuz zeichnet.

```
 10 REM : SINUS-KURVE
 20 PRINT " ( CLR/HOME ) "
 30 COLOR 0,0,7,7
 40 FOR N=0 TO 1022
 50 M=512-400* SIN (N/512 * 3.14)
 60 GRAPHIC 2
 70 POINT 1,M,N
 80 NEXT N
 90 DRAW 2, 0, 512TO1023, 512TO1023, 512
100 DRAW 2, 512, 512TO512, 1023TO512, 0
```

Versuchen Sie nun mal, daraus ein Programm zu erstellen, das es Ihnen ermöglicht, eine beliebige Funktion einzugeben. Eine kleine Hilfe am Rand: sehen Sie sich die Zeilen 40 und 50 an und überlegen Sie einmal, ob da nicht etwas mit Variablen zu machen ist.

Das nächste Programm nennt sich Kaleidoskop, nicht ohne Grund, wie Sie nach dem Eingeben feststellen werden. Viel Spaß !!

```
  1 REM : KALEIDOSKOP
  5 PRINT " (CLR/HOME) "
 10 GRAHPIC 2 : COLOR 6,0,5,5
 20 REGION 5
 50 DEF FNA (X) = INT (RND (1)*X) +1
100 FOR X=1TO1023 STEP FNA (70)+10
105 COLOR 2,0,5,5
110 DRAW 1, X, 0TO1023, X
120 DRAW 1, 0, XTOX, 1023
130 DRAW 1, X, 0TO0, 1023-X
140 DRAW 1, 1023, XTO1023-X, 1023
145 COLOR 0,0,5,5
150 DRAW 1, X, 0TO1023-X, 1023
160 DRAW 1, 1023, XTO0, 1023-X
180 IF X=1020 THEN210
200 NEXT
210 FOR I=1TO4500 : NEXT
```

```
220 GRAPHIC 4 : COLOR 1,3,6,6
230 PRINT " NOCH EINMAL ??? (J/N) "
240 GET ZZ$
260 IF ZZ$= "N" THEN PRINT :GOTO300
270 IF ZZ$ <> "J" THEN 240
280 GOTO 10
300 FOR I=1TO2000 : NEXT
310 END
```

Das folgende Programm simuliert eine 3 D-Darstellung. Sie werden nach dem Abstand gefragt, der die Variable "D" hat. In Zeile 4 wird die Variable umbenannt nach "P" und mit dem Faktor 2,7 addiert und taucht in der Zeile 110 wieder auf. Um den Abstand der Kreise zu berechnen, benötigt der Computer eine weitere Variable, die hier ebenfalls mit "D" benannt wird (Zeilen 6+7). Nachdem der Computer die Pseudo 3 D-Darstellung gezeichnet hat, wird der Bildschirm in Zeile 156 gelöscht und der Computer beginnt mit dem Zeichnen der vorher eingegebenen Kreise. In Zeile 230 wird wieder in den normalen Graphik-Bildschirm zurückgeschaltet. Dieses geschieht mit dem Befehl "GRAPHIC 4" und dem Color-Befehl "COLOR 1,3,6,6". Sie werden aufgefordert, ein neues Bild nach Eingabe Ihrer Koordinaten zeichnen zu lassen. Beantworten Sie diese Aufforderung mit nein, springt der Computer zum Ende des Programms und zeigt Ihnen den noch frei verfügbaren Speicherplatz an.

```
 0 PRINT "(CLR/HOME)"
 2 PRINT "WELCHEN ABSTAND(10-90)"
 3 INPUT D
 4 P=D+2.7
 5 PRINT"WELCHER ABSTAND DER KREISE (10-100)"
 6 INPUT D
 7 D=D
 9 COLOR 0,0,0,0
10 GRAPHIC 2
20 REGION 1
50 DEF FNA (X)=INT (RND(1)*X)+1
```

```
100 FOR A= 1TO4
110 FOR X= 10TO1023 STEP FNA (P) +10
120 DRAW 1,1023-X, X TO X, O+X
130 DRAW 1,X,1023-X TO O+X,X
150 NEXT X,A
151 FORI= 1TO5000: NEXT
156 SCNCLR
157 COLOR 0,0,7,7
160 CIRCLE 3, 500,500 ,D,D
170 IF D=400 THEN GOTO 220
180 D=D+20
190 GOTO160
220 FOR I= 1 TO 2000:NEXT
230 COLOR 1,3,6,6: GRAPHIC 4
240 PRINT "NEUES BILD ??"
250 GET ZZ$
260 IF ZZ$= "N" THEN PRINT: GOTO 295
270 IFZZ$ <> "J" THEN 250
290 GOTO0
295 PRINT "(CLR/HOME)"
300 PRINT FRE (0) : PRINT " BYTES FREI"
```

In dem nächsten Programm wird ein gegebener Kreis gezeichnet, wie Sie aus Zeile 70 ersehen koennen. In Zeile 20 ist eine Abfrage, die Sie nur mit "K" oder "H" beantworten können. Solange Sie keine Eingabe gemacht haben, bleibt der Rechner in Zeile 30 stehen. In den Zeilen 40 und 50 ersehen Sie dann, in welche Zeilen der Rechner springt. Nachdem der Programmablauf beendet ist, werden Sie in Zeile 160 gefragt, ob Sie einen neuen Durchmesser wünschen. Antworten Sie mit nein, springt der Computer direkt an das Ende des Programms, dieses können Sie der Zeile 180 entnehmen. Antworten Sie positiv, können Sie einen neuen Radius wählen, der mit der Variable "D" definiert wird. Dieses "D" taucht in den Zeilen 290 und 340 wieder auf. Als nächstes können Sie eine neue Farbe wählen. Sie ist hier mit der Variablen "F" definiert. Dieses "F" finden Sie in den Zeilen 300 und 350 wieder. Der Computer zeichnet dann, je nach Ihrer

Eingabe, den angegebenen Kreis, füllt diesen oder den Hintergrund aus. Danach wird der Rechner angewiesen, in eine Warteschleife zu gehen und erneut nach einem neuen Durchmesser zu fragen. Sie können die Position des Kreises dadurch ändern, indem Sie eine neue Abfrage hinzufügen und sie mit der Variablen "P" kennzeichnen. Dieses "P" müßte dann in den Zeilen 290 , 340 und 360 durch den angegebenen Wert 500 ersetzt werden.

```
 10 PRINT "(CLR/HOME)"
 20 PRINT  "SOLL DER KREIS ODER DER  HINTERGRUND
             AUSGEFüLLT WERDEN (K/H)"
 30 GET ZZ$
 40 IF ZZ$="K" THEN PRINT: GOTO 100
 50 IF ZZ$<>"H" GOTO30
 60 GRAPHIC 2: COLOR 0,0,5,7
 70 CIRCLE 2,500,500,200,200
 80 REGION 2
 90 PAINT 2,0,0
 95 GOTO 140
100 GRAPHIC 2: COLOR 0,0,5,7
110 CIRCLE 2,500,500,200,200
120 REGION 2
130 PAINT 2,500,500
140 FORI= 1TO3000:NEXT
150 GRAPHIC 4 : COLOR 1,3,6,6
155 SCNCLR
160 PRINT "NEUER DURCHMESSER (J/N)??"
170 GET ZZ$
180 IF ZZ$="N" THEN PRINT:GOTO 500
190 IF ZZ$<>"J" THEN 170
200 PRINT "(CLR/HOME)"
210 PRINT "WIE GROSS (10-450)?"
220 INPUT D
221 PRINT "NEUE FARBE (1-159)?"
222 INPUT F
230 PRINT"SOLL DER KREIS ODER DER HINTERGRUND
             AUSGEFüLLT WERDEN (K/H)?"
```

```
240 GET ZZ$
260 IF ZZ$="K" THEN PRINT: GOTO 330
270 IF ZZ$<>"H" GOTO 240
280 GRAPHIC 2 :COLOR 0,0,5,7
290 CIRCLE 2,500,500,D,D
300 REGION F
310 PAINT 1,500,1023
320 GOTO 370
330 GRAPHIC 2: COLOR 0,0,5,7
340 CIRCLE 2,500,500,D,D
350 REGION F
360 PAINT 2,500,500
370 FOR I= 1TO3000 :NEXT
380 GOTO 150
500 END
```

Als nächstes folgt ein Programm, das nach Eingabe der Variablen "A", die für den Abstand steht, eine Aneinanderreihung von mehreren Kreisen, die in sich selber eine geschlossene Ellipse darstellen. Die Pokes in den Zeilen 1,2 und 8 verkleinern den Bildschirm auf den maximal benötigten Platz, um die Fragen in den Zeilen 6 + 9 darzustellen.

Die Anweisung in der Zeile 120 veranlaßt den Computer, die Umrandung der Darstellungsfläche zu zeichnen. Nachdem dieses geschehen ist, wird er in den Zeilen 140 und 150 angewiesen, die Kreise zu berechnen, wobei hier wieder die Variable "A" auftaucht, die den Abstand der Kreise angibt, dann wird der Kreis, der mit der Variablen "X", die ja vorher in der Zeile 50 ausgerechnet wurde und unserer bereits am Anfang eingegebener Variable "D" besteht. Sobalt die Funktion "ARC" 6.3 ergibt, geht der Rechner zum Anfang des Programmes zurück. Ist dies nicht der Fall, wird er in der Zeile 170 angewiesen, einen neuen Kreis zu zeichnen, so lange, bis der Wert 6.3 erreicht ist.

```
  1 POKE 36883,10
  3 POKE 36881,25
  4 PRINT "(CLR/HOME)"
  5 PRINT "WELCHER ABSTAND (.10-.90)??"
  6 INPUT A
  7 PRINT "(CLR/HOME)"
  8 POKE 36883,10: POKE 36881,120
  9 PRINT "WELCHEN  URCHMESSER (10-250)??"
 10 INPUT D
 30 GRAPHIC 2: COLOR 0,0,13,7
120 DRAW 2,0,0TO0,1023TO1023,1023TO1023,0TO0,0
140 FOR ARC = 0 TO 2.0 * 3.14 STEP A
150 X=512 +200* COS (ARC) :Y=512+200 * SIN (ARC)
160 CIRCLE 2,X,Y,D,D
165 IF ARC= 6.3 THEN 175
170 NEXT ARC
175 FORT=1 TO 4000: NEXT
180 GRAPHIC 4 : COLOR 1,3,6,6
190 GOTO1
```

4.) Der VC-20 Sound

4.1 Grundlagen

Wenn Sie die übersicht über die Register des Video-Chip aufmerksam studiert haben, so konnten Sie feststellen, daß fünf Register dieses elektronischen Bausteins für die Tonerzeugung zuständig sind. Der Name "Video-Chip" ist für dieses kleine technische Wunderwerk also etwas untertrieben.
Die fünf Soundregister des Video-Chip sind die Register a bis e, d. h. die Register 10 bis 14. Wie Sie bereits wissen, liegt die Startadresse des Videochip bei 36864. Daraus ergibt sich, daß die Speicherplätze für die fünf Soundregister die Speicherstellen 36864+10=36874 bis 36878 sind.
Wie sind nun die Aufgaben unter diesen fünf Registern verteilt? Die ersten drei Register sind normale Tongeneratoren. Sie können Töne über fünf Oktaven erzeugen. Das vierte Register ist der Rauschgenerator. Es dient zur Erzeugung von Geräuschen und Geräuscheffekten. Das fünfte Register schließlich kontrolliert die Lautstärke .
Um einen Ton oder ein Geräusch zu erzeugen, müssen Sie immer zwei Dinge tun: Einmal die Lautstärke setzen und zum zweiten einen der ersten vier Register mit einem Wert größer als 127 belegen. Warum größer als 127? Um dies zu verstehen, müssen wir uns noch einmal den Plan der Video-Chip-Register ansehen. Betrachten Sie das höchstwertigste Bit der ersten vier Soundregister. Sie sehen, daß dort S1 bis S4 steht. Dies bedeutet, daß das Video-Chip so konstruiert ist, daß ein Ton erst erklingt, wenn dieses oberste Bit gesetzt ist. Dies hat den Vorteil, daß ein Tonkanal abgeschaltet werden kann, ohne daß der gespielte Ton verloren geht. Probieren Sie einmal folgendes:

```
POKE36874,195
POKE36878,15
```

Nachdem Sie dies eingetippt haben laufen lassen, erklingt ein mittelhohes C. Tippen Sie nun das Folgende ein:

```
POKE36874,PEEK(36874)-128
```

Augenblicklich verstummt der Ton. Das eben eingetippte C ist aber nicht verloren gegangen. Durch die Subtraktion von 128 haben Sie lediglich das oberste Bit des Registers auf Null gesetzt, da dieses ja die Wertigkeit 128 besitzt. Addieren wir nun wieder die 128. Tippen Sie Folgendes ein.

```
POKE36874,PEEK(36874)+128
```

Sie hören nun wieder Ihr C. Ein etwas eleganterer Weg den Kanal verstummen zu lassen ist, anstatt der Subtraktion die "und" - Verknüpfung zu benutzen. Diese Operation gewährleistet auch, daß Sie keine negativen Werte bekommen, sollte das Register zufällig keinen Wert größer als 127 haben und damit bekommen Sie auch auf keinen Fall eine Fehlermeldung. Der sichere Befehl zum Verstummenlassen eines Registers lautet also:

```
POKE36874,PEEK(36874)AND127
```

Nun ist wieder alles ruhig. Wenn Sie Ihr C wieder hören wollen, benutzen wir nun die "oder" - Verknüpfung. Diese Operation ist wieder nicht nur eleganter, sie gewährleistet auch, daß Sine Werte größer als 255 bekommen, die dann ja wieder eine Fehlermeldung verursachen würden. Tippen Sie ein:

```
POKE36874,PEEK(36874)OR128
```

Ihr C ist wieder da. Dies können Sie mit jedem der ersten vier Register machen.

Das fünfte Register ist, wie bereits erwähnt, das Kontrollregister für die Lautstärke. Für die Lautstärke sind die vier unteren Bits dieses Registers zuständig. Dies bedeutet, daß Sie Werte zwischen 0 und 15 in die Speicherstelle 36878 schreiben können, um die Lautstärke zu regeln. Der Wert 0 bedeutet, daß die Lautstärke 0 ist,d.h. Sie hören nichts. 15 ist demnach die größte Laustärke.

Wir verzichten hier darauf, Ihnen eine Tabelle der Noten mit den dazugehörigen Werten für die einzelnen Speicherstellen zu geben. Eine solche Tabelle finden Sie in Ihrem VC-20 Handbuch. Stattdessen geben wir Ihnen im nächsten Kapitel ein Programm an die Hand, mit dessen Hilfe Sie sehr komfortabel Töne, Geräusche und Toneffekte erzeugen können, den Soundeditor. Weiter wollen wir Ihnen zeigen, wie Sie die Tastatur Ihres VC-20 in die eines kleinen Synthsizers verwandeln können und schließlich liefern wir Ihnen noch ein paar Anregungen, wie Sie ein Schlagzeug programmieren können.

4.2 Der Soundeditor

Wenn Sie schon einmal ein Spiel programmiert haben, haben Sie sicher auch schon vor dem Problem gestanden, daß Sie ein bestimmtes Geräusch erzeugen wollten. Sie haben dann sicher sehr lange die verschiedensten Werte eingetippt und ausprobiert. Der Soundeditor macht nun die Erzeugung von Tönen und Geräuscheffekten um vieles einfacher.

Folgende Funktionen hat der Soundeditor:

Wenn Sie das Programm starten, sehen Sie sechs Reihen auf dem Bildschirm. Die ersten vier Reihen entsprechen den ersten vier Soundregistern des Video-Chip. Die Reihen 5 und 6 haben eine besondere Funktion. Geräuscheffekte haben die Eigenschaft, einen längeren Zeitraum zu beanspruchen. Innerhalb dieses Zeitraums erreichen die Geräusche Ihre maximale Lautstärke und verklingen dann wieder. Dieser Zeitraum kann sehr kurz sein, z.B. das Geräusch der Rotorblätter eines Hubschraubers oder auch sehr lang, z.b. das Martinshorn eines Feuerwehrwagens. Die Reihen mit den Namen "Ansteigen" bzw. "Abfallen" gestatten Ihnen nun genau diese Effekte zu programmieren.

Und so arbeiten Sie mit dem Soundeditor:

Wählen Sie zuerst mit Hilfe der Cursortasten hoch und runter die gewünschte Reihe an. Die ausgewählte Reihe erscheint nun in inverser Darstellung. Wenn Sie nun den Wert dieser Reihe erhöhen wollen, so benutzen Sie die Funktionstaste F1. Zu dem Wert dieser Reihe wird eins addiert. Wollen Sie den Wert um eine größere Zahl ändern, benutzen Sie die Funktionstaste F2 (SHIFT-F1) . Zu dem Wert dieser Reihe werden nun 10 addiert. Wollen Sie den Wert der Reihe erniedrigen, so benutzen Sie die Funktionstaste F3 um den Wert um eins zu vermindern und die Taste F4 (SHIFT-F3) um den Wert um zehn zu vermindern. Für die ersten 4 Reihen werden keine Werte kleiner als 127 oder größer als 255 angenommen, da sie

nicht sinnvoll bzw. nicht erlaubt sind. Die beiden letzten Reihen dürfen Werte zwischen 1 und 999 annehmen. Wenn Sie die RETURN-Taste drücken, erklingt der Ton, der sich aus sämtlichen auf dem Bildschirm sichtbaren Werten ergibt.
Auf der nächsten Seite finden Sie das Listing zu dem Soundeditor.
Hier noch ein paar Erläuterungen zu einzelnen Programmzeilen:

Zeile 10-60 Anfangswerte, Ober- und Untergrenzen und die Repeatfunktion für alle Tasten werden gesetzt (Zeile 18).

Zeile 70 Abfrage ob Taste gedrückt

Zeile 75-76 Abfrage ob Cursortaste gedrückt

Zeile 80-92 Bewegung durch Cursortasten

Zeile 93-96 Abfrage der Funktionstasten

Zeile 100 Abfrage der RETURN-Taste

Zeile 110-142 Änderung der Werte und Ausdruck

Zeile 2000-2040 Spielen des Tons

Haben Sie nun Ihren gewünschten Toneffekt gefunden, so müssen Sie sich nur noch die Werte der 6 Zeilen vom Bildschirm abschreiben und in Ihrem Programm eine Schleife wie in den Zeile 2000-2040 einbauen. In Zeile 2000 werden die Werte der ersten vier Reihen in die entsprechenden Speicherstellen geschrieben. Die restlichen Zeilen sind für das Ansteigen und Abfallen des Tones verantwortlich.
Sie sehen also, die Erzeugung eines Effektes ist um vieles einfacher geworden. Auf den nächsten beiden Seiten finden Sie nun das Listing des Programms. Wir wünschen Ihnen viel Spaß bei der Entdeckungsreise durch die verschiedensten Toneffekte.

```
5 REM **** SOUNDEDITOR ****
10 V=36878:S(0)=36874:S(1)=36875:S(2)=36876:S(3)=36877
16 FORI=0TO3:O(I)=255:NEXTI:O(4)=999:O(5)=999
17 FORI=0TO3:U(I)=127:NEXTI:U(4)=1:U(5)=1
18 POKE650,128
20 PRINTCHR$(147)
30 FORI=0TO5
40 READA$(I)
50 W(I)=127
55 PRINTA$(I);W(I):PRINT
56 NEXTI
57 Z=0
60 PRINTCHR$(19):PRINTCHR$(18);A$(Z);W(Z);CHR$(146)
70 GETA$:IFA$=""THEN70
75 IFA$=CHR$(17)ANDZ<5THEN80
76 IFA$=CHR$(145)ANDZ>0THEN90
77 GOTO93
80 PRINTCHR$(145);A$(Z);W(Z):PRINT
81 Z=Z+1
82 PRINTCHR$(18);A$(Z);W(Z);CHR$(146)
85 GOTO999
90 PRINTCHR$(145);A$(Z);W(Z);
91 FORI=1TO3:PRINTCHR$(145);:NEXTI:PRINT
92 Z=Z-1:PRINTCHR$(18);A$(Z);W(Z);CHR$(146):GOTO999
93 IFA$=CHR$(133)ANDW(Z)<O(Z)THEN110
94 IFA$=CHR$(134)ANDW(Z)>U(Z)THEN120
95 IFA$=CHR$(137)ANDW(Z)<O(Z)-10THEN130
96 IFA$=CHR$(138)ANDW(Z)>U(Z)+10THEN140
100 IFASC(A$)=13THENGOSUB2000:GOTO999
105 GOTO999
110 W(Z)=W(Z)+1
111 PRINTCHR$(145);CHR$(18);A$(Z);W(Z);CHR$(146)
112 GOTO999
```

```
120 W(Z)=W(Z)-1
121 PRINTCHR$(145);CHR$(18);A$(Z);W(Z);CHR$(146);CHR$(157)" "
122 GOTO999
130 W(Z)=W(Z)+10
131 PRINTCHR$(145);CHR$(18);A$(Z);W(Z);CHR$(146)
132 GOTO999
140 W(Z)=W(Z)-10
141 PRINTCHR$(145);CHR$(18);A$(Z);W(Z);CHR$(146);CHR$(157)" "
142 GOTO999
999 GOTO70
1000 DATA "KANAL 1     ","KANAL 2     ","KANAL 3     ","RAUSCHEN    "
1010 DATA"ANSTEIGEN  ","ABFALLEN   "
2000 FORI=0TO3:POKE36874+I,W(I):NEXTI
2005 FORI=0TO15STEP15/W(4)
2010 POKEV,I:NEXTI
2020 FORI=15TO0STEP-15/W(5)
2030 POKEV,I:NEXTI
2040 RETURN

READY.
```

4.3 Der VC-20 als Synthesizer

Das folgende Programm gestattet es Ihnen, Ihren VC-20 als Synthesizer einzusetzen. Die Buchstabenreihe von Q bis fungiert als die weißen Reihen der Tastatur. Die schwarzen Tasten werden von der Zahlenreihe repräsentiert. Da nicht über jeder weißen eine schwarze Taste liegt, sind natürlich auch nicht alle Zahlentasten belegt. Betätigen Sie eine Taste aus diesen beiden Reihen, ertönt sofort der zugehörige Ton. Das Q entspricht dabei dem Ton C, das W dem Ton D u.s.w.

Dies alles ist nun zugegebenermaßen nicht sehr neu. Doch der Clou dieses Programms kommt ja noch.

Starten Sie das Programm. Sie sehen als erstes eine Abfrage, ob Sie in den Spiel- oder Programmiermodus möchten. Tippen Sie "S" für Spielmodus ein. Sie können nun die beiden obersten Reihen als Klaviertasten verwenden wie oben beschrieben. Verlassen Sie diesen Modus, indem Sie die Pfeiltaste links oben auf der Tastatur betätigen. Wählen Sie nun "P" für Programmiermodus. Sie sehen nun zwei Zeilen auf dem Bildschirm. Diese beiden Zeilen entsprechen den letzten beiden Zeilen des Soundeditors aus dem letzten Kapitel. Sie können hier nun den Verlauf des Tones genauso wie im Soundeditor bestimmen. Sind Sie mit dem Verlauf zufrieden, verlassen Sie den Programmiermodus wieder mit der Pfeiltaste links oben und gehen Sie wieder in den Spielmodus. Wenn Sie nun spielen, klingt dieser so, wie Sie es eben programmiert haben. Sie haben also einen kleinen aber feinen Synthesizer aus Ihrem VC-20 gemacht.

Schnell noch ein paar Erklärungen zu den Programmzeilen, bevor Sie sich ans Tippen stürzen:

Zeile 10-41 Die Anfangsvariablen werden gesetzt. In W1(I) und W2(I) werden die Zahlenwerte für die Töne, in T(I) die dazugehörigen Tastaturcodes gespeichert.

Zeile 45-50 Abfrage ob Spiel- oder Programmiermodus

Zeile 51-60 Initialisierung Programmiermodus

Zeile 70 Abfrage Taste gedrückt

Zeile 75-76 Abfrage Cursortaste

Zeile 80-92 Ausführen der Cursortasten

Zeile 93-96 Abfrage der Funktionstasten

Zeile 100 Abfrage der RETURN-Taste

Zeile 101 Abfrage der Pfeiltaste

Zeile 110-141 Ausführen der Funktionstasten und Ausdruck der neuen Werte

Zeile 1000-1048 Werte für die Töne

Zeile 1050-1065 Werte für die Tastaturabfrage

Zeile 2000-2040 Spielen eines Tones

Zeile 3000 Initialisierung Spielmodus

Zeile 3010 Abfrage Taste gedrückt

Zeile 3015 wenn Pfeil links dann Spielmodus verlassen

Zeile 3020 Suchen der gedrückten Taste

Zeile 3030 nicht gefunden dann zurück zu Abfrage Taste gedrückt

Zeile 3040 Zuweisung der Variablen für Unterprogramm

Zeile 3050 Ton spielen , dann zurück zu Abfrage Taste gedrückt

Auf der nächsten Seite finden Sie nun das Listing zum Synthesizer. Auch wenn Sie drei Seiten Programm eintippen müssen, es ist die Mühe wert.

```
5 REM **** SYNTHESIZER *****
10 V=36878:S(0)=36874:S(1)=36875:S(2)=36875
15 DIMW1(23),W2(23),T(23)
16 O(4)=999:O(5)=999:U(4)=1:U(5)=1
30 FORI=4TO5:READA$(I):W(I)=127:NEXT:Z=4
40 FORI=1TO23:READW1(I):READW2(I):NEXTI
41 FORI=1TO23:READT(I):NEXTI
45 PRINTCHR$(147):PRINT"SPIEL- ODER"
46 PRINT"PROGRAMMIERMODUS"
47 PRINT"(S/P)"
48 GETA$:IFA$=""THEN48
49 IFA$="S"THEN3000
50 IFA$<>"P"THEN48
55 PRINTCHR$(147):FORI=4TO5
56 PRINTA$(I);W(I):PRINT
57 NEXTI
60 PRINTCHR$(19):PRINTCHR$(18);A$(Z);W(Z);CHR$(146)
70 GETA$:IFA$=""THEN70
75 IFA$=CHR$(17)ANDZ<5THEN80
76 IFA$=CHR$(145)ANDZ>3THEN90
77 GOTO93
80 PRINTCHR$(145);A$(Z);W(Z)
81 PRINT
82 Z=Z+1
83 PRINTCHR$(18);A$(Z);W(Z);CHR$(146)
84 GOTO999
90 PRINTCHR$(145);A$(Z);W(Z);
91 FORI=1TO3:PRINTCHR$(145);:NEXTI:PRINT
92 Z=Z-1:PRINTCHR$(18);A$(Z);W(Z);CHR$(146):GOTO999
93 IFA$=CHR$(133)ANDW(Z)<O(Z)THEN110
94 IFA$=CHR$(134)ANDW(Z)>U(Z)THEN120
95 IFA$=CHR$(137)ANDW(Z)<O(Z)-10THEN130
96 IFA$=CHR$(138)ANDW(Z)>U(Z)+10THEN140
100 IFASC(A$)=13THENGOSUB2000:GOTO999
101 IFASC(A$)=95THEN45
105 GOTO999
110 W(Z)=W(Z)+1
111 PRINTCHR$(145);CHR$(18);A$(Z);W(Z);CHR$(146)
112 GOTO999
120 W(Z)=W(Z)-1
121 PRINTCHR$(145);CHR$(18);A$(Z);W(Z);CHR$(146);CHR$(157)" "
122 GOTO999
```

```
130 W(Z)=W(Z)+10
131 PRINTCHR$(145);CHR$(18);A$(Z);W(Z);CHR$(146)
132 GOTO999
140 W(Z)=W(Z)-10
141 PRINTCHR$(145);CHR$(18);A$(Z);W(Z);CHR$(146);CHR$(157)" "
142 GOTO999
999 GOTO70
1000 DATA "ANSTEIGEN  ","ABFALLEN   "
1020 DATA 192,195,197,197
1025 DATA 200,200,203,203
1030 DATA 206,207,208,209
1035 DATA 211,212,214,214
1036 DATA 216,216,218,219,220,221
1040 DATA 222,223,224,224
1045 DATA 226,226,227,227
1046 DATA 229,229,231,231,232,232
1047 DATA 233,233,234,234
1048 DATA 236,236,237,237,238,238
1050 DATA 81,50,87,51
1055 DATA 69,82,53,84
1056 DATA 54,89,55,85,73,57
1060 DATA 79,48,80,64
1065 DATA 45,42,92,94,19
2000 FORI=0TO3:POKE36874+I,W(I):NEXTI
2005 FORI=0TO15STEP15/W(4)
2010 POKEV,I:NEXTI
2020 FORI=15TO0STEP-15/W(5)
2030 POKEV,I:NEXTI
2040 RETURN
3000 PRINTCHR$(147):PRINT"SPIELMODUS"
3010 GETA$:IFA$=""THEN3010
3015 IFASC(A$)=95THEN45
3020 FORI=1TO23:IFASC(A$)=T(I)THEN3040
3030 NEXTI:GOTO3010
3040 W(1)=W1(I):W(2)=W2(I):W(3)=0:W(0)=0
3050 GOSUB2000:GOTO3010

READY.
```

4.4 Schlagzeug auf dem VC-20

Ein besonders faszinierendes Gebiet der Geräuscherzeugung ist die Imitierung eines Schlagzeuges. Auch wenn Ihre Anverwandten Ihre Versuche in dieser Richtung vielleicht als störend empfinden könnten, sollte Sie das nicht abschrecken. Leider geht es aber auch hier nicht ohne ein bißchen Theorie.

Bevor Sie sich an die Programmierung eines Schlagzeuges geben, müssen Sie sich zuerst einmal die verschiedenen möglichen Klangkörper eines Schlagzeuges vor Augen führen und herausfinden, wie Sie diese erzeugen. Hier hilft nur langes Probieren (und die kleinen Beispiele als Hilfe am Ende dieses Kapitels). Eine nützliche Hilfe wird Ihnen dabei sicher der Soundeditor sein. Wenn Sie den Klang Ihrer Trommeln herausgefunden haben, können Sie an die eigentliche Programmierung gehen. Sie überlegen sich die Taktart, wo die Betonungen liegen und legen entsprechend Ihre Instrumente. Wir können hier und jetzt leider keinen kompletten Kursus über Rhythmik und verschiedene Taktarten abhalten. Wenden Sie sich dazu bitte an einen Schlagzeuger oder die einschlägige Fachliteratur. Wir wollen Ihnen nur ein Beispiel geben, wie Sie einen 3/4 Takt erzeugen können. Die Auswahl der Trommeln ist relativ willkürlich und Sie müssen Sie Ihrem eigenen Geschmack anpassen.

Das folgende Programm erzeugt also einen Walzertakt. In Zeile 10 werden die Variablen für die Register gesetzt, in Zeile 20 die Lautstärke auf maximales Volumen gesetzt. In Zeile 1000 wird das erste Instrument gespielt. Der erzeugte Ton ist relativ tief (Baßtrommel) und betont den ersten Taktschlag sehr gut. In Zeile 1005 wird eine Warteschleife aufgerufen, die die Geschwindigkeit des Rhythmus bestimmt. Nun folgen zwei weniger betonte Taktschläge (Snare). Dies geschieht in den Zeilen 1010 bis 1060. Zeile 1998 bewirkt, daß der nächste Takt begonnen wird. Sie

sehen, daß es gar nicht allzu schwer ist, einen Rhythmus zu programmieren.

Hier nun das Listing:

```
10 V=36878:S(0)=36874:S(1)=36875:S(2)=36876:S(3)=36877
20 POKEV,15
1000 POKES(0),128
1001 POKES(1),139
1002 POKES(1),0
1003 POKES(0),0
1005 GOSUB2000:REM WARTESCHLEIFE
1010 FORI=1TO2
1020 POKES(2),222
1030 POKES(3),244
1040 POKES(2),0
1050 POKES(3),0
1055 GOSUB2000:REM WARTESCHLEIFE
1060 NEXTI
1998 GOTO1000
2000 FORW=1TO500:NEXTW
2010 RETURN
```

Auf der folgenden Seite finden Sie nun noch ein paar Beispiele für verschiedene Klänge eines Schlagzeuges.
Wir vereinbaren folgende Werte für die Variablen:

```
S(1)=36874:S(2)=36875:S(3)=36876:S(4)=36877:V=36878
```

Zuerst noch einmal die Werte aus unserem Walzer-Rhythmus:

```
POKEV,15
POKES(1),128
POKES(2),138
POKES(1),0
POKES(2),0
```

Der hieraus resultierende Ton ist der einer Basstrommel und gut für Betonungen geeignet.

Der zweite Ton, der einer Snare, ist folgender:

```
POKEV,15
POKES(1),221
POKES(2),244
POKES(1),0
POKES(2),0
```

S-Tom und H-Tom lassen sich mit den folgenden zwei Kombinatioen imitieren:

```
L-Tom
POKEV,15
POKES(3),165
POKES(3),166
POKES(3),167
POKES(3),0
```

```
H-Tom
POKEV,15
POKES(3),194
```

```
POKES(3),195
POKES(3),193
POKES(3),0
```

Das Hi-Hat wird so erzeugt:

```
POKEV,15
POKEV,12
POKES(4),254
POKEV,6
POKES(4),254
POKES(4),0
```

Das Stockschlagen des Schlagzeugers schließlich können Sie so imitieren:

```
POKEV,12
POKES(3),245`
POKES(3),244
POKES(3),245
POKES(3),0
```

Sie haben nun die wichtigsten Instrumente eines Schlagzeugers zur Hand. Der Karriere Ihres VC-20 als Rhythmusmaschine steht eigentlich nichts mehr im Wege.

KAPITEL 5 : POKES UND ANDERE NÜTZLICHE ROUTINEN

In diesem Kapitel werden Sie eine ganze Menge nützliche Routinen kennenlernen, die Sie in Ihren Programmen verwenden können, um diese schneller und eleganter zu machen. Es empfiehlt sich in diesem Zusammenhang, sich außerdem noch mit dem Betriebssystem des VC-20 zu beschäftigen. Sie werden nicht nur einige trickreiche POKES finden, sondern auch Informationen über verschiedene PEEKS und SYS. Die meisten Routinen lassen sich sowohl von BASIC als auch von Maschinenprogrammen aufrufen. Es ist leicht verständlich, daß durch solche Routinen Ihre Programme bedeutend schneller und kürzer werden.

Zunächst aber erst etwas über die verschiedenen Befehle zum Aufruf der Routinen.

Von BASIC aus:

NEW

Ein Befehl, der unter Umständen fatale Folgen haben kann. Trotzdem sollte auch seine Wirkungsweise hier besprochen werden. Wenn Sie ein Maschinenprogramm geladen haben, sollten Sie danach jedesmal den NEW-Befehl anwenden. Er setzt die wichtigen Register in der Zero-Page auf ihren Urzustand zurück. So können Sie auch zusätzlich noch ein BASIC Programm eingeben, ohne schon nach der ersten Eingabe eine eigenartige Fehlermeldung zu bekommen. Natürlich gehen bei diesem Befehl alle BASIC Programme, die zu diesem Zeitpunkt im Speicher stehen verloren - MASCHINENPROGRAMME ABER NICHT !

PEEK

Eine sehr interessante Funktion ist die PEEK-Funktion. Mit dieser Funktion können Sie alle Speicherstellen auslesen. Verändert wird hierbei überhaupt nichts. Sie sollten aber darauf achten, daß die

Resultate dieser Funktion in dezimaler Form ausgegeben werden. Um nun diese Resulate genau zu verstehen, kann es manchmal nützlich sein, sie in hexadezimale Form zu übersetzen. überhaupt spielt diese Form der Zahlendarstellung in diesem Bereich der Programmierung eine große Rolle. Ein Beispiel der PEEK-Anwendung soll dieses verdeutlichen:

Um den Anfang des BASIC-Bereiches zu bekommen, also den BASIC-Programmstart, können Sie folgende Eingabe machen:

```
PRINT PEEK (43) + 256 * PEEK (44)
```

Das Resultat lautet zum Beispiel bei einem ohne Speichererweiterung ausgerüstetem VC-20 4097, bei einem Gerät mit der 3K Supererweiterung 1025 und bei einer 8K oder 16K Erweiterung 4609.

Nun noch etwas zu der Darstellung der Rechnung. PEEK (43) ist das untere Byte der 2 Byte Adresse und PEEK (44) das obere Byte. Wir sprechen in diesem Zusammenhang auch von Low- und Highbyte. Um das richtige Resultat zu erlangen, müssen wir das Highbyte noch mit 256 multiplizieren, und dann noch das Lowbyte addieren. Entsprechend lassen sich auch alle anderen Adressen berechnen, der BASIC-Programmstart soll hier nur als Beispiel diesen.

Weitere interesannte Adressen der Zeropage sind noch:

	Adressen
Zeiger auf den Start der Variablen	(45, 46)
Zeiger auf den Start der Arrays (Felder)	(47, 48)
Zeiger auf das Ende der Arrays	(49, 50)
Zeiger auf den Start der Strings (Texte)	(51, 52)
Zeiger auf das Ende des BASIC-RAM	(55, 56)

Zu der Benutzung dieser Adressen kommen wir später noch.

POKE

Durch den POKE-Befehl werden die einzelnen Speicherstellen mit dem angegebenen Wert überschrieben. Auch hier ist darauf zu achten, daß die Wert in dezimaler Form eingegeben werden. Außerdem dürfen sie nur im Bereich von 0 bis 255 liegen. Andere Werte würden zu einem ILLIGAL QUANTITY ERROR führen. Doch passen Sie bei der Benutzung des POKE-Befehls besonders auf: Sie können sich damit Ihre ganzen Programme zerstören, oder das Gerät ganz zum "Absturz" bringen. Vor allen Dingen BASIC-Programme sind bei der Benutzung sehr gefährdet. überlegen Sie sich also genau, in welches Bienennest Sie "hineinpoken".

SYS

Dieser Befehl ruft ein Maschinenprogramm oder eine Maschinenroutine auf. Interessant sind in diesem Zusammenhang die möglichenen Anwendungen des SYS-Befehls. So können Sie nicht nur Programme, durch Angabe der Startadresse starten, sondern auch noch Werte an das Maschinenprogramm übergeben. Denken wir zum Beispiel an ein Graphikprogramm, so könnte man sich vorstellen, eine Gerade durch Angabe von von Anfangs- und Endkoordiante zu zeichnen. Umgesetzt auf den SYS-Befehl könnte das so aussehen:

```
SYS 6144,0,100
```

Wir gehen hierbei davon aus, daß das Maschinenprogramm, bzw. die entsprechende Routine, an der Stelle 6144 (oder hex 1800) beginnt, die Startkoordiante 0 und die Endkoordinate 100 ist. Der Computer versteht zunächst nur die Startadresse des Maschinenprogramms. Die anderen Paramter, in unserem Fall also 0 und 100, werden in verschiedenen Registern bereit gestellt. Vom Programm aus können dann die verschiedenen Werte verarbeitet werden. An folgenden Adressen können Sie die Inhalte der Register finden:

	Adresse
Akku für den SYS-Befehl	780
X-Register für den SYS-Befehl	781
Y-Register für den SYS-Befehl	782
Status-Register für den SYS-Befehl	783

Bei unserem obigen Beispiel, würde also an der Stelle 780 der Wert 0 und an der Stelle 781 der Wert 100 gespeichert.

Auch dieser Befehl kann seine Tücken haben. So startet ein

SYS 64802

den VC-20 wieder - das Programm ist weg !!!

USR

Auch mit dieser Funktion können Sie ein Maschinenprogramm oder eine Maschinenroutine aufrufen. Sie müssen dabei aber ein paar Sachen berücksichtigen. Es gibt 2 besondere Adressen in der Zeropage des VC-20, die die Benutzung von USR steuern. Diese beiden Adressen sind die Speicherplätze 1 und 2. Diese beiden Stellen arbeiten zusammen mit der Adresse 0. In dieser Adresse steht der Wert 76. Er bedeutet JMP (also springe) im Sinne der Maschinensprache. Danach folgt die Adresse, zu der gesprungen werden soll. Auch hier müssen wir also wieder in Low- und Highbyte trennen. Adresse 1 ist das Low- und Adresse 2 das Highbyte der Maschinenroutine, an die, bei Aufruf des USR-Befehls, verzweigt werden soll. Mit Hilfe des USR-Befehls ist es zum Beispiel möglich, eigene mathematische Routinen zu entwickeln, und dann den Wert oder die Werte berechenen zu lassen. Ein solcher Vorgang könnte zum Beispiel so aussehen:

```
10 POKE 1,0: REM LSB
20 POKE 2,24: REM MSB
```

```
30 A=10
40 B=USR(A)
50 PRINT "SQR VON ";A;" IST ";B
60 END
```

ACHTUNG: Dieses Programm soll nur die Anwendung der USR-Funktion verdeutlichen. Das Programm ist so, wie es hier als reines BASIC-Programm steht, nicht lauffähig. Es wird davon ausgegangen, das ein Maschinenprogramm, welches die Wurzel einer Zahl berechnet, bei der Adresse 6144 (oder hex 1800) beginnt. In die Speicherstellen 1 und 2 werden also Low- (0) und Highbyte (24 = hex 18) geschrieben. Durch Aufruf der USR-Funktion verzweigt der Computer dann in das Maschinenprogramm und übergibt der Variablen B den Wert der Wurzel.

WAIT

Dies ist der letzte Befehl, auf den wir im Zusammenhang mit dem Aufruf von Maschinenprogrammen und Routinen hinweisen wollen. Er ist vielleicht nicht ganz so einfach zu verstehen, wie die anderen Befehle. Aber trotzdem kann er sehr nützlich sein.

Mit Hilfe dieses Befehles kann der Zustand einer Speicherstelle abgefragt werden. übersetzt würde der Befehl ungefähr bedeuten: Warte solange, bis an der Adresse X das Signal Y anliegt. Genauer gesagt: WAIT 654,1 wartet solange, bis die SHIFT-Taste gedrückt wird. Werden andere Zeichen eingegeben, werden diese zwar in den Tastaturpuffer übernommen aber nicht eher angezeigt, wie die SHIFT-Taste gedrückt wird. Der Grund dafür ist, daß diese Stelle normalerweise den Wert 0 enthält - solange die SHIFT-Taste nicht gedrückt ist. Wird diese Taste jedoch gedrückt, dann wird der Wert dieser Adresse zu 1, und damit die WAIT Schleife abgeschlossen.

Sie müssen nicht alle Befehle des VC-20 genaustens beherrschen.

Es ist oftmals aber sehr nützlich, über Ihren Zweck informiert zu sein, um so zum Beipsiel leichter Programme von anderen Rechnern übertragen zu können. Um es noch einmal zu sagen: Seien Sie, solange Sie Ihren VC-20 nicht absolut kennen, vorsichtig im Umgang mit den oben angesprochenen Befehlen. Es könnte sonst zu unangenehmen Überraschungen kommen.

5.1 Routinen des Betriebssystems und ihre Anwendung

Wie Sie oben sicherlich schon bemerkt haben, gibt es verschiedene Routinen, die bei der Programmierung, sowohl von BASIC als auch von der Maschinensprache sehr nützlich sein können. Wir wollen Ihnen hier einen Liste der wichtigsten Routinen geben. Für eine umfassende Auflistung aller Routinen, empfehlen wir Ihnen die Lektüre des VC-20 ROM-Listings.

Hier nun die Adressen:

Adresse: (mögliche) Anwendung:

0000-0002 X=PEEK(1)+256*PEEK(2)

An dieser Stelle finden Sie die Adressen für die Anwendung der USR-Routine. 0000 hat normalerweise den Wert 76, das der Opcode für JMP bedeutet. Adresse 0001 ist das Low- und Adresse 0002 das Highbyte der zu benutzenden USR-Routine.

0020-0021 X=PEEK(20)+256*PEEK(21)

An dieser Stelle steht die Adresse, an der BASIC die Integer-Variablen speichert, die in dem BASIC-Programm benutz werden.

0043-0044 X=PPEK(43)+256*PEEK(44)

Diese Adresse zeigt auf den Anfang des Benutzerspeichers, also auf den Start des BASIC-Programms. Wie Sie diese Adresse auswerten können, haben wir Ihnen schon oben ausführlich beschrieben.

0045-0046 X=PPEK(45)+256*PEEK(46)

In dieser Adresse finden Sie den Start des Speicherplatzes, bei dem die numerischen Variablen beginnen. Dieser Adresse liegt für gewöhnlich direkt hinter dem BASIC-Programmspeicher.

0047-0048 X=PEEK(47)+256*PEEK(48)

Hier steht die Anfangsadressse für den Array-Speicherplatz. Alle Felder (Arrays) des BASIC-Programms werden in diesem Speicherplatz abgelegt.

0049-0050 X=PEEK(49)+256*PEEK(50)

Diese Adresse deutet auf das Ende des Array-Speicherplatzes.

0051-0052 X=PEEK(51)+256*PEEK(52)

Bei dieser Adresse beginnen die Strings, also Textvariablen, des BASIC-Programmes.

0055-0056 X=PEEK(55)+256*PEEK(56)

Zeiger auf das Ende des BASIC-RAMs. Durch Versetzen dieser Adresse, ist es möglich, einen bestimmten Speicherbereich vor überschreiben zu schützen. Da sich die anderen Adressen an dieser Adresse orientieren, können Sie einen ganzen Teil des Speichers "sperren". Dieser Teil könnte nun von einem Maschinenprogramm aus genutzt werden. Zum Beipiel die mathematische Routine für die USR-Funktion in unserem Beispiel. So könnte eine Sperrung erfolgen:

Bei Standard VC-20:	Nach POKE 56,20:
Adresse 55: Wert 0	Adresse 55: Wert 0
Adresse 56: Wert 30	Adresse 56: Wert 20
FRE (0) : 3581	FRE (0) : 1021
RAM-Ende : 7680	RAM-Ende : 5120

0097-0102 X=PEEK(97), Y=PEEK(98) ...

Akkumulator 1 für Fließkommazahlen. (FAC) Die Adresse 102 stellt das Vorzeichen der entsprechenden Zahl dar.

0105-0110 X=PEEK(105), Y=PEEK(106) ...

Akkumualtor 2 Für Fliekommazahlen. (ARG) Die Adresse 110 stellt das Vorzeichen der entsprechenden Zahl dar.

0115-0138 MASCHINENSPRACHE: JSR $0073

Eine sehr wichtige Routine. Sie holt das nächste Zeichen aus dem BASIC-Text.

0122-0123 X=PEEK(122)+256*PEEK(123)

Dieser Zeiger zeigt auf die aktuelle, zu verarbeitende, Zeile.

0144 X=PEEK(144)

Die Variable ST, also das Statuswort, ist hier gespeichert. Bei der Abfrage von ST wird der Wert von dieser Adresse aus gelesen.

0152 X=PEEK(152)

Die Anzahl der offenen Dateien ist an dieser Stelle abgelegt.

0153 X=PEEK(153)

Die IEC-Adresse für das aktuelle Eingabegerät. Normalerweise steht hier 0 - bedeutet Tastatur.

0154 X=PEEK(154)

Die IEC-Adresse für das aktuelle Ausgabegerät. Normalerweise steht hier 3 - bedeutet Bildschirm.

0157 X=PEEK(157)

Im Direktmodus, also über Tastatur, steht an dieser Stelle der Wert 128. Bei Ablauf eines Programmes steht hier eine 0. Es wird also signalisiert in welchem Modus sich der Computer gerade befindet.

0160-0162 X=PEEK(160), Y=PEEK(161) ...

An dieser Stelle werden die Zeitvariablen TI und TI$ abgelegt. Bei der BASIC-Abfrage dieser Variablen, holt sich der Computer die aktuelle Zeit aus diesen Adressen.

0174-0175 X=PEEK(174)+256*PEEK(175)

Diese Adresse zeigt auf das Programmende bei einem LOAD oder SAVE Befehl. Um also zum Beispiel die Länge eines Programmes nach dem Laden zu berechnen, müssen Sie nur die Differenz zwischen dem BASIC-Programmstart (43, 44) und dieser Adresse bilden.

0183-0186 X=PEEK(183), Y=PEEK(184) ...

In der ersten Speicherstelle steht die Länge des Programmnamens, welches geladen oder gespeichert werden soll, und in der zweiten Adresse, die logische Filenummer. Die dritte Adresse beinhaltet dann noch die Sekundäradresse und die letzte die Gerätenummer. Diese Adressen sind vor allen Dingen bei der Maschinenprogrammierung interessant, um ein Programm, oder Daten, zu laden oder zu speichern.

0187-0188 X=PEEK(187)+256*PEEK(188)

Auch diese Adresse gehört zu einem Lade- oder Speichervorgang. Sie zeigt nämlich auf die Adresse, unter der der Programmname gespeichert ist.

0197 X=PEEK(197)

Eine sehr interessante Adresse für das Einlesen von Zeichen über die Tastatur. In dieser Speicherstelle steht solange der Wert 64, bis eine Taste gedrückt wird.

0198 X=PEEK(198)

In dieser Speicherstelle wird die momentane Länge des Tastaturpuffers abgespeichert.

0199 X=PEEK(199)

Flagge ob RVS-Modus eingeschaltet ist oder nicht. Bei eingeschaltetem RVS-Modus ist das Resultat 1, bei ausgeschaltetem Modus 0.

0201-0202 X=PEEK(201), Y=PEEK(202)

In der Speicherstelle 201 ist die Cursorzeile und in 202 die Cursorspalte zwischengespeichert, an welcher die Eingabe erfolgen soll. Somit ist also auch auf einfache Art und Weise eine Cursorpositionierung möglich.

0206 X=PEEK(206)

An diesere Stelle steht der Wert des Zeichens, das sich gerade unter dem Cursor befindet.

0209-0210 X=PEEK(209)+256*PEEK(210)

Dieser Zeiger weist auf den Start der aktuellen Bildschirmzeile.

0211 X=PEEK(211)

In dieser Speicherstelle kann die momentane Position des Cursors innerhalb der Zeile ausgelesen werden.

0213 X=PEEK(213)

Diese Adresse enthält die Anzahl der Bildschirmzeilen einer Zeile. Wie Sie wissen, kann die Eingabe einer Zeile ja insgesamt über maximal 4 Bildschirmzeilen erfolgen. In dieser Adresse können also die Werte 21, 43, 65 oder 87 stehen (0-21 ergibt 22 Zeichen per Zeile).

0214 X=PEEK(214)

Diese Adresse ähnelt sehr stark der Adresse 211, mit dem Unterschied, daß in dieser Speicherstelle die momentane Bildschirmzeilennummer festegehalten wird.

Um den Cursor an eine bestimmte Postion zu setzen, müssen also insgesamt 4 Adressen, nämlich 201, 210, 211 und 214 verändert werden.

0216 X=PEEK(216)

Diese Adresse spielt bei der Verwendung des Insert-Modus eine große Rolle. Um zu verhindern, daß mehr als die erlaubten 88 Zeichen eingegeben (eingefügt) werden, steht in dieser Adresse die restliche Anzahl der Zeichen, die noch eingefügt werden können. Bei einem POKE 216,0 wird der Insert-Modus ausgeschaltet.

0217-0240 X=PEEK(217), Y=PEEK(218) ...

Eine sehr interessante Adresse für die Bearbeitung einer BASIC-Zeile (oder auch einer normalen Eingabe) ist der Bereich von 217 bis 240. In diesem Bereich wird eine ganze Bildschirmzeile zwischengespeichert. Wie Sie feststellen, ist in diesem Bereich Platz für insgesamt 23 Zeilen; 22 Zeichen sind aber nur erlaubt. Der Grund dafür ist, daß das letzte Zeichen für die Erkennung eines Zeilenendes benutzt wird. Ist dieses (23te) Zeichen 30, so bedeutet das, daß die Zeile fortgesetzt wird. 158 bedeutet dagegen, daß die Zeile nun zu Ende ist.

0243-0244   X=PEEK(243)+256*PEEK(244)

Dieser Zeiger deutet auf das Farb-RAM.

0251-0254

Diese Adressen sind eigentlich für den Maschinenprogrammierer interessant. Sie werden vom VC-20 nicht belegt, und sind somit frei für den Anwender verfügbar. Der große Vorteil der Zero-Page ist darin zu sehen, daß für die Abspeicherung von Werten (und natürlich auch beim Lesen dieser Adressen) wesentlich weniger Speicherplatz im Anwenderprogramm benötigt wird. Erinnern wir uns: Die Adressen 0-255 werden hexadezimal dargestellt als 00-FF. Alle andern Zahlen gehen von 0100-FFFF. Wenn ich jetzt in die Zero-Page schreiben will, so genügt zum Beispiel ein STA $FB - also ein 2-Byte Befehl. Bei anderen Adressen müßte ich dagegen sagen: STA $1800 - also 3 Bytes. Das klingt wenig. Kann aber trotzdem bei größeren Maschinenprogrammen von enormem Vorteil sein - außerdem geht´s so wesentlich schneller.

0512-0600   X=PEEK(512), Y=PEEK(513) ...

Dieser große Bereich dient als Eingabepuffer vom BASIC aus. Geben Sie also eine Zeile ein, wird diese im Gesamten hier zwischengespeichert.

0631-0640   X=PEEK(631), Y=PEEK(632) ...

Haben Sie sich nicht schon oft gewundert, warum verschiedene Programme, die von Kassette geladen werden, automatisch starten ? Der Grund für dieses "Phänomen" ist in diesen Adressen zu suchen. Sie enthalten den Tastaturpuffer. Jede Eingabe eines Zeichens, daß Sie über die Tastatur aus eingeben, wird an dieser Stelle gespeichert. Ist der Platz belegt, werden die Zeichen in den Eingabepuffer übertragen, und es können weitere Zeichen eingegeben werden. Diese übertragung geschieht so schnell, daß Sie dieses gar nicht bemerken. Der Trick des Autostartes ist nun

einfach der, daß diese Adressen nicht von der Tastatur aus gefüllt werden, sondern vom Programm. Es werden also die normalen Befehle (z.B. RUN) in diese Speicherstelle geschrieben, und durch das Zeichen 13 (für RETURN) aus gestartet. Dazu muß nur noch die Adresse 198 geändert werden (s.o.).

0641-0642 X=PEEK(641)+256*PEEK(642)

Diese Adresse zeigt auf den Start des Speichers.

0643-0644 X=PEEK(643)+256*PEEK(644)

Diese Adresse zeigt auf das Ende des Speichers.

0646 X=PEEK(646)

In dieser Adresse wird der momentan benutzte Farb-Code für die Zeichenfarbe gespeichert.

0648 X=PEEK(648)

Um festzustellen, bei welcher Adresse der Video-RAM beginnt, können Sie diese Adresse lesen. Da das Low-Byte immer gleich ist, braucht hier nur das High-Byte angesprochen zu werden.

0649 X=PEEK(649)

Diese Adresse steuert die Länge des Tastatutpuffers. Normalerweise ist die Länge dieses Puffers mit 10 angesetzt (siehe Adressen 631 bis 640). Diese Länge können Sie nun für eigene Zwecke verkürzen. Eine Erweiterung kann verschiedene andere Adressen zerstören.

0650 X=PEEK(650)

Wie Sie wissen, sind nicht alle Tastaten "wiederholungsfähig". Das heißt, daß bei längerem Drücken der einzelnen Taste, das

Zeichen nicht wiederholt wird (wie zum Beispiel die Pfeil-Tasten). Durch Veränderung des Wertes von 0 auf 128 werden nicht nur die Cursor-Steuer-Funktionen, sondern auch alle anderen Tasten wiederholt.

0651 X=PEEK(651)

Durch Veränderung dieser Adresse können Sie den Zeitfaktor, nach dem die Wiederholung des Zeichens beginnt (s. o.), verändern. Dadurch können Sie verhindern, daß bei einer unbeabsichtigten Verzögerung des Tastendrucks, das Zeichen nicht einfach wiederholt wird - der Zeitfaktor wird also länger eingestellt. Oder aber Sie wollen, daß die Wiederholung des Zeichens schon früher einsetzt - der Zeitfaktor wird also verkürzt. Sie müssen etwas experimentiern, um die beste Zeitverzögerung zu erreichen.

0653 X=PEEK(653)

Mit dieser Adresse können Sie die SHIFT und CTRL- Taste sperren. Das kann vorteilhaft sein, um zu verhindern, daß der Anwender bei verschiedenen Programmen flasche Eingabe macht. Wenn Sie also Ihre Programme vervollkommnen wollen, so sind es gerade diese Feinheiten, die den "Normalprogrammierer" vom "Experten" unterscheiden.

0654 X=PEEK(654)

Mit dieser Adresse können Sie den Zustand der SHIFT-Taste abfragen.

0657 X=PEEK(657)

Wenn Sie bei der Adresse 653 die Tasten SHIFT und CTRL steuern können, so besteht nun die Möglichkeit die SHIFT und C= Tasten zu sperren. Damit können Sie verhindern, daß bei der Benutzung eines Programmes, unbeabsichtigt vom Groß/Klein- in Groß/Graphik-Modus geschaltet werden kann.

0768-0769 X=PEEK(768)+256*PEEK(769)

Diese Adresse zeigt auf die Fehlertabelle des VC-20. Wenn Sie also nun deutsche Fehlermeldungen wollen, so müssen Sie diese nur an einen bestimmten Speicherplatz schreiben (schützen nicht vergessen!) und diese Adresse verändern. Natürlich müssen die Fehlercodes übereinstimmen. Das heißt, das der Fehler mit dem Code x auch nach der Umsetzung zum Fehlercode x gehört.

0828-1019 X=PEEK(828), Y=PEEK(829) ...

In diesem Speicherbereich befindet sich der Puffer des Kassettenrecorders. Alle Informationen, die von der Kassette geladen werden, werden also hier abgespeichert. Wenn Sie nicht mit der Kassette arbeiten wollen, so können Sie diesen Speicherplatz für Maschinenprogramme nutzen. Sobald Sie aber Aus- oder Eingaben über den Kassettenrecorder amchen, wird der ganze Bereich überschrieben.

5.2 KERNAL

Einführung

über diesen besonderen Teil des VC-20 Betriebssystems wird von allen Seiten viel zu wenig berichtet - wo er doch einer der wichtigsten und interessantesten Teile des ganzen ROMs darstellt. Wir wollen an dieser Stelle nun ausführlich die Anwendung des Kernals und seinen eigentlichen Zweck beschreiben.

Kernal - Was ist das ?

Um es kurz zu sagen: Kernal ist eine sogeanannte "Sprungtabelle" für verschiedene Funktionen des VC-20 Betriebssystems. Es handelt sich hierbei um Ein- und Ausgaberoutinen, und um die Routinen zur Speicherverwaltung des VC-20.

Würde nun, aus irgendwelchen Gründen, das VC-20 Betriebssystem verändert, und sei es nur um ein Byte, kann es natürlich passieren, daß Ihre Maschinenprogramme nicht mehr ordnungsgemäß ablaufen - sofern Sie sich direkt auf Routinen ausserhalb des Kernals beziehen. Dagegen bleiben die Sprungadressen im VC-20 Kernal unverändert. Wenn Sie also zum Beispiel die Routine BSOUT aufrufen wollen, die ein Zeichen aus dem Akku an das angeschlossene Ausgabegerät sendet, so können Sie direkt sagen:

JMP $FFD2 oder von BASIC SYS 65490

An der Stelle 65490 steht nun normalerweise ein Sprung zu der Adresse, die unter 806 bis 807 gespeichert ist. An dieser Stelle befindet sich der Output-Vektor. Würde die Firma nun Änderungen an dieser Adressierung unternehmen, so ist es für Sie uninteressant, ob die Adresse 65490 an die Adresse 806,807 verweist oder an irgendeine andere Stelle. Denn Sie brauchen nur die Kernal Routine aufzurufen - der Rest geschieht automatisch. Selbst das Kernal könnte geändert werden - nur die Einsprungadressen dürfen nicht geändert werden. Deren Wert spielt

dagegen gar keine Rolle.

Es werden vom Kernal insgesamt 39 Routinen, wie sie oben besprochen wurden unterstützt. Wenn Sie diese Routinen anstelle der direkten Sprünge in die Routine anwenden, werden Sie keine Schwierigkeiten mit der Umsetzung auf einen anderen Rechner mehr haben und dabei noch sehr viel Zeit sparen. Außerdem ist dieses Programm dann auch für denjenigen, der es nicht kennt, viel einfacher zu "lesen" und gegebenenfalls zu ändern.

Um die verschiedenen Funktionen anwenden zu können, müssen Sie sich an ein paar Regeln halten. Zunächst müssen Sie die verschiedenen Register mit den benötigten Parametern laden und dann die Funktion aufrufen. Beachten müssen Sie dabei allerdings, daß einige Register bei dem Durchlauf des Betriebssystems verändert. Bevor Sie also die Kernal-Routine aufrufen, sollten Sie sich sicher sein, daß nicht unbeabsichtigt verschiedene Register, die Sie in Ihrem Programm benötigen, zerstört werden. Wir wollen Ihnen in diesem Kapitel ausführlich die Anwendung der Kernal-Routinen aufzeigen. Anwenden müssen Sie diese dann selber.

Jede Kernal-Routine hat verschiedene Teile. Auf diese Teile wollen wir nun etwas genauer eingehen, da wir sie in unserer Beschreibung jeder einzelnen Kernal-Routine noch benötigen werden. Sollten Sie sich nicht genau an diese "Benutzeranleitung" der Kernal-Routinen halten, kann es recht leicht passieren, daß Ihr Programm diesen Aufruf nicht "überlebt".

1.) Der Name der Funktion

Dieser Name spielt eigentlich in der Anwendung dieser Routinen keine größere Rolle. Wir haben aber versucht, durch leicht verständliche Namen (z.B. LISTEN, BASIN, BSOUT etc.) auch eine einprägsame Anwendung zu ermöglichen.

2.) Die Einsprungadresse der Funktion

Ein sehr wichtige Teil innerhalb der Kernal-Funktion ist

natürlich die Einsprungadresse. Wie wir oben schon aufgezeigt haben, ist es später unerheblich, auf welche Adresse die Routine verweist. Wichtig ist nur diese Einsprungadresse zu kennen.

3.) Die übergaberegister an die Funktion

Wir müssen der Kernal-Routine mitteilen was wir eigentlich wollen. Also zum Beispiel welchen Kanal wir öfnnen wollen, oder aber welches Zeichen wir senden wollen. Für diese Mitteilungen stehen uns die verschiedenen Register zur Verfügung (Akku, X-Register, Y-Register, Status, Programmzähler, Stack Pointer).

4.) Kernal-Routinen, die vor dieser Routine aufgerufen werden müssen

Einige Kernal-Routinen benötigen Daten, die in anderen Routinen erarbeitet bzw. umgewandelt werden. Zu diesem Zweck führen wir hier auch noch diese Routinen auf, die vor der eigentlichen Kernal-Routine aufgerufen werden müssen.

5.) Die Rückgabe von Fehlermeldungen ans Maschinenprogramm

Trifft die Kernal während der Verarbeitung auf einen Fehler, so wird dies dem Maschinenprogramm, aus dem sie aufgerufen wurde gemeldet. Zu diesem Zweck wird zunächst das Carry-Flag gesetzt, das nach jedem Kernal-Aufruf überprüft werden sollte (z.B. BCS, BCC). Sollte ein Fehler aufgetreten sein, so steht im Akku sein Wert. Darauf kommen wir später noch. Mit Hife dieser Routine, kann man verhindern, daß das System irgendwelche seltsame Wege geht, ohne daß der Anwender darauf eingehen kann.

6.) Wieviel Stack-Platz wird benötigt ?

Die nächste wichtige Information, ist die Tiefe des Stacks, der von der Kernal-Routine benötigt wird. Diese Information benötigen Sie, um nach Aufruf der Routine, wieder an die Daten zu kommen, die Sie vorher in den Stack geschrieben haben.

7.) Von der Kernal-Routine benutzte Register

Intern benötigt die Kernal-Routine natürlich auch noch verschiedene Register. Diese Register werden hier angeführt, damit Sie sehen, welche Register vor Aufruf der Routine auf den Stapel bringen sollten, damit diese nicht überschrieben werden.

Kernal dient aber nicht nur dem Programmierer zum Aufruf verschiedener Betriebssystem-Routinen. Es überwacht bei Start des Systems noch einige andere Aufgaben.

Zunächst ist da die Abfrage, ob ein Modul verwendet wird. Haben Sie also vor Einschalten des VC-20 ein ROM-Modul eingeschoben, so erkennt Kernal dieses und gibt die Steuerung des VC-20 an dieses Modul ab. Daher ist es auch wichtig, das Modul vor Einschalten einzustecken, da sonst nicht nur der VC-20 oder das Modul beschädigt werden können, sondern auch das Vorhandensein eines Moduls gar nicht erkannt wird. Liegt nach Einschalten kein Modul vor, so geht Kernal zum nächsten Schritt.

Bei diesem Schritt überprüft und initialisiert Kernal den Speicher des VC-20. Hierbei werden zuerst die Speicherbereiche von 0 bis 255 und von 512 bis 1023 gelöscht und der Kassettenpuffer nach 828 gelgt. Somit ist dieser Bereich des Systems "startklar". Danach wird der gesamte RAM-Bereich beginnend bei 1024 getestet. Kernal sucht nun den Anfang des RAM-Bereiches, bei dem mit der Programmierung begonnen werden kann. Ist diese Adresse größer als 4096 (1024 bis 4095 ist der Platz für die 3K Erweiterung und 4096 das erste Byte des normalen RAM-Bereichs), so bedeutet das, daß das System nicht einwandfrei arbeitet (keine 3K Erweiterung und kein normaler RAM-Bereich). Aus diesem Grund wird nach einem fehlerhaften Test ein recht eigenartiger Bildschirm angezeigt. An für sich brauchen Sie sich nicht vor einem solchen Fehler fürchten, da die RAM-Bausteine sehr zuverlässig sind. Sollte aber trotzdem irgendwann mal ein solcher Fehler auftreten, so wissen Sie nun, um was es sich

handelt.

Nach diesem ersten Speichertest, bei dem der Anfang des RAMs gesucht wurde, geht Kernal nun auf die Suche nach dem Ende des RAM-Bereichs. Sollte dieses Ende kleiner als 8192 sein (bis dahin reicht der Bildschirmspeicher), so bedeutet das, daß dieser Speicherbereich fehlerhaft ist, und der VC-20 antwortet wie oben beschrieben.

Danach folgt ein weiterer, wichtiger Schritt. Wie Sie sicherlich wissen, verändert sich ja bei Speichererweiterung teilweise auch der Bildschirmspeicher. Wenn das RAM-Ende größer als 8448 ist, dann ist der Start des Bildschirmspeichers bei 4096, der Speicheranfang liegt bei 4608 und reicht bis zum errechneten RAM-Ende. Sollte das Ende des RAMs jedoch kleiner als 8448 sein, so liegt der Bildschirmspeicher bei 7680 und das Ende des RAMs liegt ebenfalls bei 7680. Nun können Sie auch erkennen, warum eigenartuge Zeichen auf dem Bildschirm erscheinen, wenn Sie versuchen ein Programm zu laden, das nicht in den vorhandenen Speicherplatz paßt.

Zum Schluß werden noch alle anderen Vektoren gesetzt, die die Ein- Ausgabesteuerung übernehmen, die Sprungtabelle von 788 bis 819 wird gebildet, die oben schon angesprochene CHRGET Routine wird abgespeichert und die erste Bildschirmmeldung ausgegeben:

```
*** CBM BASIV V2 ***

3583 BYTES FREE

READY.
```

Nun ist Ihr VC-20 bereit Ihre Befehle entgegenzunehmen. Hätten Sie gedacht, daß schon, bevor Sie nur ein Zeichen auf dem Bildschirm zu sehen ist, mit dem VC-20 so viel passiert ? Wir hoffen, daß Ihen nun einige Zusammenhänge, von denen man bisher keine Ahnung hatte klarer geworden sind, und Sie Ihren VC-20 nun

besser "verstehen".

Auf den nächsten Seiten wollen wir nun im einzelnen auf die Routinen des Kernal eingehen. Es emphiehlt sich in dem Zusammenhang mit dem VC-20 ROM-Listing zusammen zu arbeiten, damit wirklich alle Zusammenhänge, zwischen Aufruf der Kernal-Routine und deren Ausführung, klar werden.

Funktion: INPUT
Einsprung: $FFA5 65445
übergabereg.: Akku
Routinen: TALK, SATALK
Fehler: s. STATUS
Stacktiefe: 13
Register: Akku, X-Register

Diese Kernal-Funktion holt Daten vom Seriellen Bus (IEC-Bus). Diese Daten können entweder von der Floppy, dem Kassettenrecorder oder einem anderen angeschlossenen IEC-Gerät. Diese Daten werden dann in den Akkumulator geschrieben, von wo sie aus dem Maschinenprogramm entnommen werden können. Bevor diese Routine aufgerufen werden kann, ist darauf zu achten, daß mit Hilfe der TALK-Routine erst der entsprechende Kanal für den Datentransfer geöffnet wurde. Falls das angeschlossene Gerät eine zweite Adresse benötigt, so muß auch noch die SATALK-Routine aufgerufen werden. Falls während dem Datentransfer ein Fehler aufgetreten ist, so kann der mit Hilfe der STATUS-Funktion ausgewertet werden.

Beispiel:

```
JSR  $FFA5
STA  $1800,X
```

Funktion: CHKIN
Einsprung: $FFC6 65478
übergabereg.: X-Register
Routinen: OPEN
Fehler: 3,5,6
Stacktiefe: -

Register: Akku, X-Register

Um aus einer Datei Informationen zu lesen, muß diese Datei zum Lesen geöffnet werden. Diese Aufgabe übernimmt dieses Register. Dabei muß die Datei zwar schon mit Hilfe der OPEN-Routine geöffnet sein, sie erhält aber durch die CHKIN die Anweisung zum Lesen. Dieses öffnen entfällt dann, wenn von der Tastatur aus gelesen werden soll. Ansosnsten gibt das X-Register die logische Filenummer an.

Beispiel:

```
LDX  #$02
JSR  $FFC6
```

Funktion: CHKOUT
Einsprung: $FFC9   65481
übergabereg.: X-Register
Routinen: OPEN
Fehler: 3,5,7
Stacktiefe: -
Register: Akku, X-Register

Diese Routine hat den umgekehrten Zweck der CHKIN. Nachdem die Datei geöffnet wurde (nicht nötig bei Ausgabe auf Bildschirm), kann Sie zum Schreiben vorbereitet werden. Auch hier enthält das X-Register die logische Filenummer.

Beispiel:

```
LDX  #$03
JSR  $FFC9
```

Funktion: BASIN
Einsprung: $FFCF   65487
übergabereg.: Akku
Routinen: OPEN, CHKIN
Fehler: s. STATUS
Stacktiefe: -
Register: Akku, X-Register

Mit dieser Routine können Sie über den mit den Kernal-Routine CHKIN und der OPEN-Routine geöffneten Datenleitung, Daten in den VC-20 Einlesen. So können zum Beispiel auch längere Text von Tastatur aus eingelesen werden, genau so wie Daten von Band oder Floppy. Beim Einlesen von Tastatur, werden die Zeichen einzelnen in den Input-Puffer geschrieben, von wo sie danach weiter verarbeitet werden können.

```
Beispiel:           JSR  $FFCF
                    STA  $1800,X
```

Funktion: BSOUT
Einsprung: $FFD2 65490
übergabereg.: Akku
Routinen: OPEN, CHKOUT
Fehler: s. STATUS
Stacktiefe: -
Register: -

Diese Funktion arbeitet im Prinzip genau so, wie die BASIN-Routine, mit dem Unterschied, daß die Zeichen nicht eingelesen sondern ausgegeben werden. Dazu muß wieder die Datei geöffnet werden (entfällt bei Bildschirmausgabe) und zum Schreiben vorbereitet werden. Sie müssen aber darauf achten, daß nur die Dateien geöffnet sind, auf die der Text, oder die Zeichen, ausgegeben werden sollen. Denn die BSOUT-Routine schreibt auf alle offene Dateien - also aufgepaßt !

```
Beispiel:           LDX  #$04      ;Druckerkanal
                    JSR  $FFC9     ;öffnen
                    LDA  #$41      ;'A'
                    JSR  $FFD2     ;ausgeben
```

Funktion: OUTPUT
Einsprung: $FFA8 65448
übergabereg.: Akku
Routinen: LISTEN, SALISTEN

Fehler: s. STATUS
Stacktiefe. -
Register. Akku

Diese Routine schickt ein Zeichen an den IEC-Bus. Dieser Bus muß zuvor mit der LISTEN-Funktion, und eventuell, falls eine Sekundäradresse angegeben werden muß, noch durch die SALISTEN-Funktion bezeichnet werden. Danach kann Zeichen für Zeichen über den seriellen Bus geschickt werden.

```
Beispiel:          LDA  #$41      ;'A'
                   JSR  $FFA8     ;senden
```

Funktion: CLALL
Einsprung: $FFE7 65511
übergabereg.: -
Routinen: -
Fehler: -
Stacktiefe: 11
Register: Akku, X-Register

Diese Routine schließt alle offenen Dateien, und modifiziert die entsprechenden Vektoren.

```
Beispiel:          JSR  $FFE7
```

Funktion: CLOSE
Einsprung: $FFC3 65475
übergabereg.: Akku
Routinen: -
Fehler: -
Stacktiefe: -
Register: Akku, X-Register

Mit Hilfe dieser Routine können Sie einzelne, logische Dateien schließen, die vorher mit der OPEN-Routine geöffnet wurden. Im Akku steht die Adresse der zu schließenden Datei.

Beispiel: LDA #$04 ;CLOSE 4
JSR $FFC3

Funktion: CLRCH
Einsprung: $FFCC 65484
übergabereg.: -
Routinen: -
Fehler: -
Stacktiefe: 9
Register: Akku, X-Register

Diese Routine schließt alle offenen Kanäle. Diese kann zum Beispiel nach einem Lesen von Diskette oder Kassette nützlich sein. Ansonsten würden die nächsten Befehle auf alle angeschlossenen und geöffneten Kanäle wirken - und wer will, wenn nicht tatsächlich beabsichtigt, die Floppy als Drucker benutzen ?

Beispiel: JSR $FFCC
Funktion: GETIN
Einsprung: $FFE4 65512
übergabereg.: Akku
Routinen: -
Fehler: -
Stacktiefe: -
Register: Akku, X-Register

Diese Routine ließt ein Zeichen vom Tastaturpuffer (entfernt ihn von dort) und speichert seinen ASCII-Wert im Akku. Sollte kein Zeichen im Puffer gewesen sein, so ist der Wert des Akkus 0.

```
Beispiel:       A1    JSR  $FFE4     ;warte bis eine Teste gedrückt
                      CMP  #$00
                      BEQ  A1
```

Funktion: IOBASE
Einsprung: $FFF3 65523

Übergabereg.: X-Register, Y-Register
Routinen: -
Fehler: -
Stacktiefe: 2
Register: X-Register, Y-Register

Diese Routine dient hauptsächlich dazu, die Programme von anderen CBM-Programmen (VIC) auch nach einer Änderung diverser Adressen, anwenden zu können. Wie Sie ja vielleicht wissen, ist es möglich, Programme von größeren Commodore Geräten auf dem VC-20 zu laden. BASIC Programme können so nicht nur gelistet, sondern auch verändert und nach den notwendigen Änderungen sogar gestartet werden. Bei den großen CBM ist das leider nicht so einfach möglich. Ihnen fehlt diese Routine.

Beispiel:

```
JSR $FFF3
STX $1800
STY $1801
```

Funktion: LISTEN
Einsprung: $FFB1 65457
Übergabereg.: Akku
Routinen: -
Fehler: s. STATUS
Stacktiefe: -
Register: Akku

Mit dieser Routine können Sie ein angeschlossenes IEC-Gerät zwischen 4 und 31 öffnen, damit von diesem Gerät Daten übertragen werden können. Der entsprechende Wert (4-31) muß natürlich vorher in den Akku gebracht werden.

Beispiel:

```
LDA #$08
JSR $FFB1
```

Funktion: LOAD
Einsprung: $FFD5 65493

übergabereg.: Akku, X-Register, Y-Register
Routinen: SETFLS, SETNAM
Fehler: 0, 4, 5, 8, 9
Stacktiefe: -
Register: Akku, X-Register, Y-Register

Diese Routine lädt Daten von einem beliebigen Eingabegerät (z.B. Floppy oder Kassette) direkt in den Speicher. Die Information des Akku besagt, ob geladen (LOAD) oder verglichen (VERIFY) werden soll. Im ersten Fall hat der Akku den Wert 0, im zweiten Fall den Wert ein. Soll der Vorspann nicht mitgeslen werden, um zum Beispiel das Programm (Maschinenprogramm) an eine bestimmte Stelle zu Laden, so müssen im X- und Y-Register die Anfangsadresse des Programms stehen.

Beispiel:

```
                LDA  #$08      ;LADE PROGRAMM VON DISK
                LDX  #$01
                LDY  #$01
                JSR  $FFBA
                LDA  #$04      ;LÄNGE DES NAMENS
                LDX  #$00
                LDY  #$18
                JSR  $FFBD
                LDA  #$00      ;00 BEDEUTET LADEN
                LDX  #$FF
                LDY  #$FF
                JSR  $FFD5
                STX  VARTAB
                STY  VARTAB+1
                JMP  STAR
       $1800:   .BYT 'TEST'
```

Funktion: MEMBOT
Einsprung: $FF9C 65436
übergabereg.: X-Register, Y-Register
Routinen: -
Fehler: -

Stacktiefe: -

Register: X-Register, Y-Register

Diese Routine kann entweder den Start des Speichers anzeigen oder verändern. Um zu wählen, ob gelesen oder geschrieben werden soll, dient bei dieser Funktion das Übertrag (Carry) Bit. Ist dieses Bit gesetzt, so bedeutet das, daß der Start nur gelesen, ein gesetztes Bit bedeutet, daß der Start geschrieben werden soll.

```
Beispiel:        SEC           ;Übertrag setzen
                 JSR  $FF9C
```

Funktion: MEMTOP

Einsprung: $FF99 65433

Übergabereg.: X-Register, Y-Register

Routinen: -

Fehler: -

Stacktiefe: 2

Register: X-Register, Y-Register

Durch diese Routine können Sie das Ende des Speichers lesen oder schrieben. Genau wie ber der MEMBOT-Routine, so zeigt auch hier das Übertrag-Bit an ob gelesen oder geschrieben werden soll.

```
Beispiel:        CLC           ;Übertrag löschen
                 JSR  $FF99
```

Funktion: OPEN

Einsprung: $FFC0 65472

Übergabereg.: -

Routinen: SETLFS, SETNAM

Fehler: 1, 2, 4, 5, 6

Stacktiefe: -

Register: Akku, X-Register, Y-Register

Diese Kernal-Routine öffnet eine logische Datei. Somit kann diese Datei für Ein- Ausgabeanweisungen benutzt werden. Die Routinen

SETFLS und SETNAM müssen vorher aufgerufen werden.

Beispiel: JSR $FFC0

```
Funktion:       PLOT
Einsprung:      $FFF0     65520
übergabereg.:   Akku, X-Register, Y-Register
Routinen:       -
Fehler:         -
Stacktiefe:     2
Register:       Akku, X-Register, Y-Register
```

Mit Hilfe dieser Routine, können Sie den Cursor an eine bestimmte Bildschirmposition setzen. Diese Position setzt entweder den Cursor an die Bildschirmposition, die durch die X- und Y-Register gekennzeichnet ist, indem das übertrag-Bit gelöscht ist, oder man lädt die beiden Register mit der Position, auf der der Cursor zur Zeit des Aufrufs steht - dann muß das übertrag-Bit gesetzt sein.

Bespiel:

```
LDX  #$0A      ;Cursor nach 10,2
LDY  #$02
CLC
JSR  $FFF0
```

```
Funktion:       RDTIM
Einsprung:      $FFDE     65502
übergabereg.:   Akku, X-Register, Y-Register
Routinen:       -
Fehler:         -
Stacktiefe:     2
Register:       Akku, X-Register, Y-Register
```

Diese Routine ließt die Zeit der eingebautet System-Uhr. Die Zeit wird entsprechend in den 3 Registern abgespeichert. Von dort kann Sie an eine bestimmte Speicherstelle abgelegt werden.

Beispiel:

```
JSR  $FFDE
STY  $1800
```

```
STX $1801
STY $1802
```

Funktion: STATUS
Einsprung: $FFB7 65463
übergabereg.: Akku
Routinen: -
Fehler: -
Stacktiefe: 2
Register: Akku

Mit dieser Routine können Sie den Status der Ein- Ausgabegeräte abfragen. Es wird entweder der Status des Gerätes oder aber der Fehler ausgegeben.
Folgende Werte sind möglich:

ST Bit	ST Wert	Kassette lesen	Seriell lesen / Schriben
0	01		Time out
1	02		"
2	04	Kurzer Block	
3	08	Langer Block	
4	16	Lese-Fehler	
5	32	Prüfsummen-Fehler	
6	64	EOF	EOI
7	128	EOT	Device not present

Funktion: RESTOR
Einsprung: $FF8A 65415
übergabereg.: -
Routinen: -
Fehler: -
Stacktiefe: 2
Register: Akku, X-Register, Y-Register

Durch Anwendung dieser Routine können Sie wieder alle Vektoren auf ihren alten Zustand (nach dem Start) zurückführen.

Beispiel: JSR $FF8A

Funktion: SAVE
Einsprung: $FFD8   65496
übergabereg.:   Akku, X-Register, Y-Register
Routinen: SETLFS, SETNAM
Fehler: 5, 8, 9
Stacktiefe: -
Register: Akku, X-Register, Y-Register

Ähnlich der LOAD Routine, so arbeitet auch diese SAVE Routine. Der Unterschied besteht darin, daß ein bestimmter Speicherbereich auf Kassette oder Diskette abgespeichert wird. Dazu wird die Anfangsadresse des Programms in der Zero-Page zwischengespeichert, die X- und Y-Register mit dieser Adresse geladen und dann die Routine aufgerufen.

Funktion: SCNKEY
Einsprung: $FF9F   65439
übergabereg.:   -
Routinen: -
Fehler: -
Stacktiefe: -
Register: Akku, X-Register, Y-Register

Diese Routine überprüft den Zustand der Tastatur danch, ob eine Taste gedrückt ist.

```
Beispiel:        A1   JSR   $FF9F
                      JSR   $FFE4
                      CMP   #$00        ;keine Taste gedrückt
                      BEQ   A1
                      JSR   $FFD2       ;gebe Zeichen aus
```

Funktion: SCREEN
Einsprung: $FFED   65517
übergabereg.:   X-Register, Y-Register

Routinen: -
Fehler: -
Stacktiefe: 2
Register: X-Register, Y-Register

Diese Routine dient hauptsächlich der Aufwärtskompatibilität. Sie stellt das Bildschirmformat fest. Beim VC-20 ist dies ja bekanntlich 22 (X-Register) und 23 (Y-Register). So können Sie zum Beispiel bei einem Maschinenprogramm feststellen, ob ein 40 Zeichen Bildschirm (z.B. CBM 64) vorliegt.

```
Beispiel:          JSR  $FFED
```

Funktion: SECOND
Einsprung: $FF93 65427
übergabereg.: Akku
Routinen: LISTEN
Fehler: s. STATUS
Stacktiefe: -
Register: Akku

Diese Routine übergibt bei Bedarf die Sekundär-Adresse an das Ein- Ausgabegerät. Diese Adresse muß vor Aufruf dieser Routine in den Akku geladen werden. Natürlich muß mit LISTEN das Gerät, an welches die Sekundär-Adresse geschickt werden soll, erst dem Computer zugewiesen werden.

```
Beispiel:          LDA  #$04
                   JSR  $FFB1
                   LDA  #$01
                   ORA  #$60
                   JSR  $FF93
```

Funktion: SETLFS
Einsprung: $FFBA 65466
übergabereg.: Akku, X-Register, Y-Register
Routinen: -

Fehler: -

Stacktiefe: 2

Register: -

Diese Routine verwaltet die logische Dateinummer (Akku), die Geräteadresse (X-Register) und die Sekundäradresse (Y-Register) für die anderen Kernal-Routinen.

Geräteadressen können sein:

Adresse	Gerät
0	Tastatur
1	Kassettenrecorder
2	RS-232C Gerät
3	Bildschirm
4	Drucker
8	Diskettengerät (seriell)

Beispiel:

```
LDA  #$01
LDX  #$08
LDY  #$01
JSR  $FFBA
```

Funktion: SETMSG

Einsprung: $FF90 65424

Übergabereg.: Akku

Routinen: -

Fehler: -

Stacktiefe: 2

Register: Akku

Mit dieser Routine können Sie die Ausgabe von Fehler- oder Kontrollmeldungen steuern. Bit 6 und 7 steuern die Herkunft der Meldung. Ist Bit 7 gesetz, wird eine Fehlermeldung aus dem Kernal gesendet. Wenn Bit 6 gesetzt ist, wird eine einfache Meldung ausgegeben.

Beispiel:

```
LDA  #$80       ;Fehlermeldung
```

```
        JSR   $FF90
```

Funktion: SETNAM
Einsprung: $FFBD 65469
übergabereg.: Akku, X-Register, Y-Register
Routinen: -
Fehler: -
Stacktiefe: -
Register: -

Diese Kernal-Routine steuert die Dateinamen zum Laden oder Speichern auf Kassette oder Diskette. Im Akku steht die Länge des Programmnamens, im X-Register das Low-Byte und im Y-Register das High-Byte der Adresse, bei der der Programmname abgespeichert ist.

Beispiel:

```
        LDA   #$04      ;Länge des Namens (z.B. TEST)
        LDX   #$00      ;ab Adresse $1800
        LDY   #$18
        JSR   $FFBD
```

Funktion: SETTIM
Einsprung: $FFDB 65499
übergabereg.: Akku, X-Register, Y-Register
Routinen: -
Fehler: -
Stacktiefe: 2
Register: -

Mit dieser Routine kann die System-Uhr gesteuert werden. Die Zeit wird in folgender Form berechnet: Alle 1/60 Sek. wir die Uhr auf den neuen Stand gebracht. Das ergibt in 24 Stunden 5.184.000 Takte. Um nun die genau Zeit einzugeben, muß der Akku mit dem ersten High-Byte, das X-Register mit dem zweiten High-Byte und das Y-Register mit dem Low-Byte der Zeit geladen werden.

Beispiel:

```
        LDA   #$00      ;00:01:00
```

```
LDX  #$00
LDY  #$3C
JSR  $FFDB
```

Funktion: SETTMO
Einsprung: $FFA2 65442
übergabereg.: Akku
Routinen: -
Fehler: -
Stacktiefe: 2
Register: -

Time-out's sind eine wichtige "Einrichtung" bei dem Datentransfer über den seriellen Bus. Durch Setzen oder Löschen von Bit 7 ist es möglich Time-out ein- (Bit 7 = 0) oder auszuschalten (Bit 7 = 1).

Beispiel:

```
LDA  #$00      ;einschalten des Time-out's
JSR  $FFA2
```

Funktion: STOP
Einsprung: $FFE1 65505
übergabereg.: Akku
Routinen: -
Fehler: -
Stacktiefe: -
Register: Akku, X-Register

Mit Hilfe dieser Kernal-Routine, können Sie jede einzelne Taste abfragen. Ist die STOP-Taste des VC-20 gedrückt, so wird die Null-Flagge gesetzt. Ansonsten enthält der Akku den Wert der letzten Tastatur-Abfrage.

Beispiel:

```
JSR  $FFE1
BNE  WEITER    ;STOP-Taste nicht gedrückt
JMP  STOP
```

Funktion: TALK
Einsprung: $FFB4 65460
übergabereg.: Akku
Routinen: -
Fehler: s. STATUS
Stacktiefe: -
Register: Akku

Mit Hilfe dieser Routine können Sie, das mit dem Inhalt des Akkus angesprochene Gerät, zum Senden öffnen. Der Inhalt des Akku kann im Bereich von 4 bis 30 liegen.

Beispiel:
```
LDA #$04
JSR $FFB4
```

Funktion: SATALK
Einsprung: $FF96 65430
übergabereg.: Akku
Routinen: TALK
Fehler: s. STATUS
Stacktiefe: -
Register: Akku

Wenn das unter TALK angesprochene Gerät zusätzlich noch eine Sekundär-Adresse benötigt, so kann diese mit dieser Kernal-Routine übergeben werden. Diese Adresse wird durch den Akku dem Gerät übergeben - der Wert darf hiebei zwischen 4 und 31 liegen.

Beispiel:
```
LDA #$04
JSR $FFB4
LDA #$04
JSR $FF96
```

Funktion: UDTIM
Einsprung: $FFEA 65514
übergabereg.: -
Routinen: -

Fehler: -

Stacktiefe: 2

Register: Akku, X-Register

Diese Routine wird normalerweise vom Kernal-Interrupt automatisch aufgerufen. Nur wenn der Benutzer eine eigene Interruptsteuerung verwendet, muß diese Funktion aufgerufen werden.

Beispiel: JSR $FFEA

Funktion: UNLSN

Einsprung: $FFAE 65454

Übergabereg.: -

Routinen: -

Fehler: s. STATUS

Stacktiefe: -

Register: Akku

Durch diese Routine werden alle Geräte angewiesen, ihren Datenempfang vom VC-20 abzubrechen. Natürlich werden nur Geräte angesprochen, die vorher mit der LISTEN-Routine geöffnet wurden.

Beispiel: JSR $FFAE

Funktion: UNTLK

Einsprung: $FFAB 65451

Übergabereg.: -

Routinen: -

Fehler: s. STATUS

Stacktiefe: -

Register: Akku

So wie die UNLSN Routine alle LISTEN-Geräte zurücksetzt, so werden alle durch TALK angesprochenen Geräte angewiesen, ihre Übertragung zu stoppen.

Beispiel: JSR $FFAB

Funktion:	VECTOR
Einsprung:	$FF8D 65412
übergabereg.:	X-Register, Y-Register
Routinen:	-
Fehler:	-
Stacktiefe:	2
Register:	Akku, X-Register, Y-Register

Mit Hilfe dieser Routine können Sie die Sprungtabelle in den Speicher übertragen und von dort wieder in den Speicherbereich zurückschreiben. So ist es Ihnen leicht möglich, die Tabelle zu modifizieren, ohne die Modifikationen direkt an den (sehr wichtigen) Normalwerten im Speicher machen zu müssen. Wenn das übertrag-Bit gesetzt ist, dann wird die Sprungtabelle an die Adresse übertragen, die durch das X- und Y-Register angegeben ist. Ist das übertrag-Bit gelöscht, so wird die Tabelle vom Benutzerspeicher wieder an den alten Speicherplatz zurückgeschrieben.

Beispiel:

```
LDX  #$00
LDY  #$18
SEC             ;lade Sprungtabelle in RAM
JSR  $FF8D
```

Fehlerkennziffern

Bei der Benutzung der Kernal-Routinen können verschiedene Fehler auftreten. Sollte dieses geschehen, wird das übertrag-Bit gesetzt und der Akku enthält die Kennziffer des Fehlers.

Wir haben Ihnen hier die vollständige Tabelle aller Fehlerkennziffern aufgeführt.

Kennziffer	Bedeutung
0	Abbruch durch Drücken der STOP-Taste
1	Zu viele offene Dateien

2	Datei bereits geöffnet
3	Datei noch nicht geöffnet
4	Datei nicht gefunden
5	Gerät nicht bereit
6	Datei ist nicht zur Eingabe geöffnet
7	Datei ist nicht zur Ausgabe geöffnet
8	Name der Datei nicht vorhanden
9	Unzulässige Gerätenummer

Wenn Sie zu diesem recht schwierigen Thema noch weitere Fragen haben, können Sie sich gerne an uns wenden. Wir stehen Ihnen gerne mit Rat und Tat zur Verfügung.

5.3 Der SYS-Befehl beim VC-20

Der SYS-Befehl dient zum Aufruf von Maschinenprogrammen von einem BASIC-Programm aus. Das Maschinenprogramm wird wie ein BASIC-Unterprogramm (GOSUB...RETURN) ausgeführt. Nach der Ausführung wird die Kontrolle wieder an das BASIC-Programm zurückgegeben. Maschinenprogramme wird man hauptsächlich dann einsetzen, wenn es auf Geschwindigkeit ankommt oder wenn Aufgaben auszuführen sind, die sich mit BASIC nur schwer oder umständlich lösen lassen. Ein typischer Anwendungsfall wäre z.B. eine Sortierroutine. In BASIC dauert das Sortieren von 100 Stringvariablen (z.B. einer Adressdatei) über eine Minute; ein entsprechendes Maschinenprogramm braucht dazu noch nicht einmal eine Sekunde. Dies sind Anwendungen für selbst geschriebene Maschinenprogramme. Aber auch Betriebssystemroutinen des VC-20 lassen sich nutzbringend anwenden. Hier bietet der SYS-Befehl des VC-20 zusätzlich noch die Möglichkeit, Parameter an das Maschinenprogramm zu übergeben und Ergebnise zurück zu bekommen.

Vier Speicherzellen des VC-20 sind für die Parameterübergabe zum und vom SYS-Befehl vorhanden. Diese Parameter werden direkt an die Register des Mikroprozessors 6502 übergeben. In Adresse 780 muß dann der Inhalt des Akkumulators stehen (das Universalregister des 6502), in Adresse 781 und 782 stehen X- und Y-Register (die Indexregister) und in Adresse 783 wird das Statusregister (die Flags) übergeben. Beim Rücksprung vom Maschinenprogramm werden in diesen Adressen die Registerinhalte wieder für BASIC zur Verfügung gestellt.

Ein Beispiel soll eine mögliche Anwendung demonstrieren.

Das BASIC des VC-20 besitzt keine Möglichkeit zur direkten Cusroradressierung. Wollen Sie zum Beispiel einen Text in Zeile 15 ab Spalte 9 ausgeben, so müssen den Cursor durch Ausgabe von Cursor Home, 15 mal Cursor down und neunmal Cursor right auf diese

Position bringen, ehe Sie den Text dort ausgeben können. Dies ist nicht nur speicherplatzaufwendig, sondern zudem auch noch langsam. Das Betriebssystem hat dafür jedoch schon eine Routine zur Verfügung. Diese Routine dient sowohl zum Setzen des Cursors als auch zum Holen der Cursorposition. Das Carryflag (Bit 0 des Statusregisters) des Prozessors dient dabei zur Unterscheidung: gelöschtes Carry bedeutet Cursor setzen, gesetztes Carryflag heißt Cursorposition holen. Die Cursorposition wird dabei im X-Register (Cursorzeile) und im Y-Register (Cursorspalte) übergeben.

Damit können wir jetzt auch von BASIC aus den Cursor direkt an jede beliebige Position des Bildschirms setzen.

```
100 REM CURSOR POSITIONIEREN
110 CU = 65520 : REM ADRESSE DER ROUTINE
120 ZE = 781   : REM CURSORZEILE IM X-REGISTER ÜBERGEBEN
130 SP = 782   : REM CURSORSPALTE IM Y-REGISTER
140 SE = 783   : REM STATUSREGISTER
200 INPUT "ZEILE, SPALTE"; Z,S
210 POKE ZE,Z: POKE SP,S: POKE SE,0 : REM CARRYFLAG LÖSCHEN
220 SYS CU : REM CURSOR SETZEN
230 PRINT "TEXT"; : REM TEXT WIRD AB DER CURSORPOSITION AUSGEGEBEN
240 GOTO 200
```

Mit dieser kleinen Routine können Sie also den Cursor ohne umständliche Stringmanipulationen an jeden beliebigen Punkt des Bildschirms setzen.

Die Routine kann ebenso zur Ermittlung der augenblicklichen Cursorposition dienen. Wenn Sie die Zeilen 100 bis 140 des obigen Programms übernehmen, sähe die Abfrage der Cursorposition so aus:

```
200 POKE SE,1 : REM CARRYFLAG SETZEN
210 SYS CU : REM POSITION HOLEN
220 Z=PEEK(ZE): S=PEEK(SP)
230 PRINT "CURSOR STAND IN ZEILE";Z;"SPALTE";S
```

5.4 Speicherung vom Maschinenprogrammen beim VC-20

Will man für den VC-20 eigene Maschinenprogramme schreiben, so stellt sich immer die Frage, wo man solche Programme abspeichern kann, damit sie nicht durch Betriebssystem oder BASIC überschrieben werden. Sie man sich die Speicherbelegung des VC-20 einmal an, so gibt es dafür mehrere Möglichkeiten.

Unbenutzte Bereiche des Betriebssystems.

Hier bietet sich in erster Linie der Puffer für Bandein- und ausgabe an. Er belegt den Adressbereich 828 bis 1019 ($033C bis $03FB). Diese 192 Bytes sind der ideale Platz für kleinere Maschinenprogramme. Aus der Nutzung als Bandpuffer ergibt sich jedoch schon eine Einschränkung: Man kann Maschinenprogramme aus diesem Bereich nicht auf Kassette abspeichern, da sie dabei mit dem Filenamen überschrieben werden. Abspeichern und Laden von Diskette ist dagegen ohne weiteres möglich. Ein weiterer ungenutzter Bereich liegt von Adresse 680 bis 767 ($02A8 bis $02FF) Dieser Bereich von 88 Bytes unterliegt keinen Einschränkungen und kann für kleine Maschinenprogramme genutzt werden; ein Abspeichern auf Kassette ist ohne weiteres möglich. Hat man längere Maschinenprogramme, muß man sich nach einem anderen Bereich umsehen. Hat man eine 8K-RAM-Erweiterung im Bereich von 40960 bis 49151 ($A000 - $BFFF), so dürfte dies auch für längste Maschinenprogramme ausreichen.

Ansonsten besteht noch die Möglichkeit, den Bereich für BASIC-Programme einzuschränken und bestimmte Bereiche für Maschinenprogramme zu reservieren. Hier kann man nun entweder den BASIC-Start nach oben verlegen oder das BASIC-Ende nach unten legen. Wie macht man das nun ?

Dazu existieren zwei Zeiger, die jeweils auf BASIC-Start und BASIC-Ende zeigen. Die Inhalte dieser Zeiger sind abhängig von der

jeweiligen Speicherkonfiguration und können von Ihnen manipuliert werden.
Dazu sehen wir uns einmal an, welche Werte die Zeiger enthalten:

```
PRINT PEEK(43) + 256*PEEK(44)
PRINT PEEK(55) + 256*PEEK(56)
```

Der erste Wert ist der BASIC-Start, der zweite gibt das BASIC-Ende an. In der Grundversion würden Sie zum Beispiel

4097 und
7680

erhalten.

Wollen wir nun Platz für ein 300-Byte-Maschinenprogramm schaffen, so können wir die 300 Byte entweder oben oder unten vom BASIC abzweigen.

Wir wollen jetzt das BASIC-Ende um 300 Bytes nach unten legen. Dazu müssen wir von dem oben erhaltenen Wert (7380) 300 abziehen - es ergibt sich 7380. Diesen Wert müssen wir nun in den Zeiger 55/56 schreiben. Dies geschieht folgendermaßen:
Der Wert muß in High- und Low-Byte zerlegt werden, was so aussieht:

```
POKE 55, 7380 AND 255
POKE 56, 7380 / 256
```

Jetzt müssen wir noch den CLR-Befehl eingeben, um die anderen BASIC-Zeiger entsprechend zu setzen. Wenn Sie jetzt

```
PRINT FRE(0)
```

eingeben, werden 300 Byte weniger ausgewiesen. Ihr Maschinenprogramm kann jetzt von Adresse 7380 bis 7679 stehen und

kann von BASIC nicht mehr erreicht werden.
Die zweite Alternative ist das Höherlegen des BASIC-Starts. Hier können wir ähnlich vorgehen wie oben.

Wir addieren jetzt zur Startadresse 4097 unsere 300 und setzen die Zeiger neu.

```
POKE 43, 4397 AND 255
POKE 44, 4397 / 256
```

Zusätzlich muß hier noch an die neue Startadresse minus 1 eine Null geschrieben werden.

```
POKE 4397-1,0
```

Die anderen Zeiger müssen mit NEW zurückgesetzt werden.

Jezt kann man das Maschinenprogramm eingeben oder auch von Kassette oder Diskette laden. Beim Laden von Maschinenprogrammen ist noch zu beachten, daß die Sekundäradresse 1 angegeben wird, da das Programm sonst an den BASIC-Start geladen wird.

```
LOAD "MASCHINENPRG", 1,1    : REM KASSETTE
LOAD "MASCHINENPRG", 8,1    : REM DISKETTE
```

5.5 Was die Maus kann, kann der Joystick schon lange

Eine interessante Amwendung für den Joystick ist z.B. eine komplette Menüsteuerung auf dem Bildschirm. Durch Bewegen des Joysticks kann der Anwender im folgenden Programm von Feld zu Feld wandern und durch Drücken des Feuerknopfs die entsprechende Funktion auslösen.

```
10 rem initialisierung
20 cu=65520 : rem cursor setzen
30 ze=781: sp=782: se=783
40 z=1: s=1
50 dd=37154: p1=37151: p2=37152: gosub1000
60 rem cusor auf mittleres feld
70 poke ze,3 : poke sp,0: poke se,0: sys cu
90 rem hauptschleife
100 poke dd,127: pa=peek(p2)
110 poke dd.254: pb=peek(p1)
120 if(pb and 4  )=0thengosub200 : rem oben
130 if(pb and 8  )=0thengosub300 : rem unten
140 if(pb and 16 )=0thengosub400 : rem links
150 if(pa and 128)=0thengosub500 : rem rechts
160 if(pb and 32 )=0then600      : rem knopf
170 print"] ";:pokeze,4*z+5:pokesp,7*s+4:pokese,0:syscu
180 print "*";: goto 100
190 rem bewegen des cursors
200 if z>0 then z=z-1
210 return
300 if z<2 then z=z+1
310 return
400 if s>0 then s=s-1
410 return
500 if s<2 then s=s+1
```

```
510 return
590 rem aktion bei feuerknopf
600 pokeze,16 : poke sp,2 :poke se,0 :sys cu
610 print "feld ";z*3+s+1 : end
990 rem felder zeichnen
1000 print"┌──────┬───────┬──────┐"
1010 print"│      │       │      │"
1020 print"│      │       │      │"
1030 print"├──────┼───────┼──────┤"
1040 print"│      │       │      │"
1050 print"│      │       │      │"
1060 print"│      │       │      │"
1070 print"├──────┼───────┼──────┤"
1080 print"│      │       │      │"
1090 print"│      │       │      │"
1100 print"│      │       │      │"
1110 print"└──────┴───────┴──────┘"
1120 return
```

Mit unserem obigen Beispielprogramm simulieren Sie etwas, was in den USA z.B. mit dem neuen Apple LISA derzeit als das Nonplusultra angeboten wird: die Maus. In unserem Programm wird wieder die Cursorsteuerung über SYS-Befehl genutzt. Anstelle der Ausgabe der Feldnummer in Zeile 610 können Sie dann Ihre gewünschten Aktionen programmieren.

5.6 Hardcopy für den VC-20

Das folgende kleine Programm erlaubt es Ihnen, jederzeit den Bildschirminhalt Ihres VC-20 auf einen angeschlossenen Drucker zu übertragen. Das kann z.B. beliebiger Text oder eine Bildschirmmaske sein. Auch Grafikzeichen werden auf den Drucker gebracht. Um die Routine so kurz wie möglich zu halten, wurde folgende Syntax gewählt:

Sie öffnen zuerst mit der logischen Filenummer eins (im Programm mit lf bezeichnet) eine Datei auf Ihrem Drucker, normalerweise mit OPEN 1,4. Dann kann mit SYS 688 eine Hardcopy zum Drucker geschickt werden. Anschließend wird der Druckkkanal wie üblich mit CLOSE 1 geschlossen.

```
130:   02b0                lf       =   1        ; logische filenummer
140:   02b0                cr       =   13       ; carriage return
170:   02b0                temp     =   $71      ; zeiger fuer ausgabe
180:   02b0                chkout   =   $ffc9    ; ausgabe auf drucker
190:   02b0                bsout    =   $ffd2    ; zeichen drucken
200:   02b0                stop     =   $ffe1    ; stoptaste abfragen
210:   02b0                store    =   $67
220:   02b0                clrch    =   $ffcc    ; ausgabe wieder auf bildschirm
300:   02b0                         *=  688      ; startadresse
350:   02b0 a9 00                   lda #0
360:   02b2 a0 1e                   ldy ##$1e    ; adresse des bildschirm $1e00
                                                 ; in der grundversion
370:   02b4 85 71                   sta temp
380:   02b6 84 72                   sty temp+1   ; zeiger auf bildschirmspeicher
420:   02b8 a6 01                   ldx lf
430:   02ba 20 c9 ff                jsr chkout   ; ausgabe auf drucker
440:   02bd a2 17                   ldx #23      ; anzahl der zeilen
450:   02bf a9 0d          loop     lda #cr
460:   02c1 20 d2 ff                jsr bsout    ; neue zeile
470:   02c4 20 e1 ff                jsr stop
480:   02c7 f0 2e                   beq exit     ; stoptaste gedrückt ?
```

```
02c9 a0 00                  ldv  #0
02cb b1 71        loop2     lda  (temp).v : zeichen vom bildschirm holen
02cd 85 67                  sta  store
02cf 29 3f                  and  #$3f
02d1 06 67                  asl  store
02d3 24 67                  bit  store    : bildschirmcode in ascci-code
02d5 10 02                  bpl  *+4      : wandeln
02d7 09 80                  ora  #$80
02d9 70 02                  bvs  *+4
02db 09 40                  ora  #$40
02dd 20 d2 ff               jsr  bsout    : und ausgeben
02e0 c8                     inv           : spaltenzahler erhohen
02e1 c0 16                  cpv  #22      : schon letzte spalte ?
02e3 d0 e6                  bne  loop2
02e5 98                     tva
02e6 18                     clc
02e7 65 71                  adc  temp
02e9 85 71                  sta  temp     : zeiger auf nächste zeile
02eb 90 02                  bcc  *+4
02ed e6 72                  inc  temp+1
02ef ca                     dex           : schon alle zeilen ?
02f0 d0 cd                  bne  loop
02f2 a9 0d                  lda  #cr      : neue zeile
02f4 20 d2 ff               jsr  bsout
02f7 4c cc ff exit          jmp  clrch    : ausgabe wieder auf bildschirm
```

5.7 Doppelt hohe Zeichendarstellung

Mit dem folgenden Programm wird eine doppelt hohe Darstellung von Zeichen auf dem Bildschirm erreicht. Doch geben Sie zuerst einmal folgenden Befehl ein : Poke 36867,47. Sie sind jetzt sicher überrascht, doch keine Sorge.Es handelt sich hier um einen Darstellungsmodus,in dem die Zeichen übereinander gezeigt werden. Wenn Sie nun eine Taste betätigen,werden Sie bemerken, daß der gedrückte Buchstabe nicht auf dem Bildschirm erscheint, sondern immer zwei Buchstaben untereinander die nicht zusammengehören. In dem folgenden Programm werden wir nun alle Zeichen richtig und auch doppelt hoch darstellen. Dieses Programm ist nur für die Grundversion des VC-20 gedacht. Wenn Sie diese Programm auf einem ausgebauten VC-20 laufen lassen wollen, müssen Sie die Anweisung in Zeile 20 entsprechend der Adresse der Erweiterung ändern.

Bitte geben Sie das Programm sehr gewissenhaft ein, denn oft sind nicht laufende Programme eine Folge fehlerhafter Eingaben. Schon eine falsche Zahl in einem POKE-Befehl zum Beispiel kann zum Ansprechen eines falscher Speicherzelle und damit zu einem völlig unbeabsichigten Resultat führen. Doch nun geben Sie bitte das folgende Programm ein und warten nach der Eingabe einen Moment!!

```
10 POKE 56,24: POKE 52,24:CLR
20 CH=32768
30 FOR X=6144 TO 7147 STEP 2
40 POKE X,PEEK (CH): POKE X+1,PEEK (CH)
50 CH=CH+1 : NEXT
55 POKE 36865,21
60 POKE 36879,25
70 POKE 36869,254
80 POKE 36867,33
90 PRINT "(CLR/HOME)"
```

So, was sagen Sie dazu. Hätten Sie gedacht, daß Ihr VC-20 in der

Lage ist, solche Sachen zu können ? Doch nun noch einmal zu dem Proramm. In Zeile 10 wird dem Computer gesagt, an das Ende des Speichers zu gehen. In den Zeilen 20-50 wird der Zeichensatz aus dem ROM in den RAM geladen. Die Zeile 60 verän dert den Bildschirm. Wie Sie sehen ist es gar nicht so schwer, mit dem VC-20 solche Manipulationen zu machen.

5.8 Wie kommen die BITS auf das Band ?

Das von Commodore verwendete Verfahren zur Kassettenaufzeichnung ist PPM, in Neudeutsch PULSE POSITION MODULATION (was ist den das??). Der Commodore-Rechner schreibt die Digitalsignale direkt, also nicht in Form zweier Tonfrequenzen, auf das Band. Hierbei sind 3 verschiedene Zeiten definiert: K = kurz (176 Mikrosekunden), M = mittel (256 Mikrosekunden), L = lang (336 Mikrosekunden). Daraus werden 3 verschiedene Kombinationen gebildet, die die folgende Bedeutung haben: LLMM = BYTE; dieser Kombination geht jedem BYTE MMKK = 1, KKMM = 0 voraus.

Das heißt, daß zum Beispiel der Buchstabe "A" auf dem Band wie folgt gespeichert wird:

LLMM	MMKK	KKMM	KKMM	KKMM	KKMM	KKMM	MMKK	KKMM	MMKK
BYTE	1	0	0	0	0	0	1	0	1
BIT NR.	0	1	2	3	4	5	6	7	PARITY ODD

Dies ergibt eine Zeitspanne von 8.96 ms für ein Zeichen.

Files werden wie folgt abgespeichert:

Programmfiles	Datenfiles
Programmkopf	Filekopf

(Start und Endadresse Filename)	(Filename)
Programmkopf (Wiederholung)	Filekopf (Wiederholung)
Programm (ein Block)	Datenblock
Programm (Wiederholung)	Datenblock (Wiederholung)
Programmfiles	Datenfiles
Endblock	Endblock

Der Kopf ist wie folgt aufgebaut:

Startadresse	00 00
Endadresse	00 00
Filename (16 Zeichen)	Filename (16 Zeichen)
Füllzeichen	Füllzeichen

Ein Block besteht aus:

ca.2 sek. Vorspann
9 BYTES Count Down ($89 $88 .. $81) beim ersten Block
($09 $08 .. $01) bei Wiederholung

Daten (bei Datenfiles 192 BYTES bei Programmen)

Prüfsumme EXOR-Prüfsumme über die gesamten Daten

Endmarkierung (LLKKKKKKKKKKKKKKKKKK)

Nachspann 0.16 ms.

Wichtig ist in diesem Zusammenhang auch, daß das Aufzeichnungsverfahren für alle Commodore-Rechner gleich ist. Abgesehen von bestimmten programmtechnischen Unterschieden lassen sich so Bänder zwischen den Commodore Computern der unterschiedlichsten Baureihen austauschen.

5.9 Daten speichern mit der Datasette

Wenn Sie nun schon ein fortgeschrittener Programmierer sind, werden Sie sicher den Wunsch haben auch Daten auf Band zu schreiben oder Daten zu lesen. Wir wollen dies nun an folgendem Beispiel durchgehen. Bitte vergessen Sie nicht, daß Daten nicht wie ein Programm eingelesen werden können, sondern immer nur in Verbindung mit einem Hauptprogramm.

Open A,B,C, "Name"

Hier wird nun ein logisches File geöffnet, wobei der Name das File identifiziert.

Die verwendeten Variablen (A,B,C) bedeuten:

A   Sucht die Hauptnummer von 1 bis 255. Wenn Ihr Programm mehr als ein File braucht, muß jedes File seine eigene Nummer haben.

B   Ist die Adresse der Datasette (=1)

C   Hier wird festgelegt, ob von der Datasette gelesen oder geschrieben wird.

Wenn C=0 wird von der Cassette gelesen.

Wenn C=1 oder 2 wird auf das Band geschrieben.

1. Daten auf Band schreiben

```
10 OPEN 1,1,1, "TEST DATEN"
20 FOR I=1 TO 10
30 PRINT # 1,1
40 NEXT
50 CLOSE 1
```

Dieses Programm eröffnet unter der logischen # 1 ein Datei mit dem Namen "TEST DATEN". Es ermöglichte Ihnen auf das Band zu schreiben. Es werden die Daten eingelesen, danach wird die Datei wieder geschlossen.

2. Lesen von Daten mit GET-Anweisungen (Einzelne Zeichen)

```
10 OPEN 1,1,0, "TEST DATEN"
20 GET #1, D$
30 IF ST AND 64 THEN GOTO 60
40 PRINT D$
50 GOTO 20
60 PRINT D$
70 CLOSE 1
```

Dieses Programm liest Daten vom Band. Es werden nur einzelne Daten eingelesen. Die Anweisung in Zeile 30 sagt dem Rechner solange Daten anzunehmen bis keine Daten mehr zur Verfügung stehen. Diese Programm ist sehr sinnvoll für diejenigen, die nur Zahlen einlesen wollen, also zum Beispiel bei einer Kontenabrechnung.

3. Lesen von Daten mit INPUT-Anweisungen (Mehrere Zeichen)

```
10 OPEN 1,1,0, "TEST FILE"
20 INPUT #1, D$
30 IF ST AND 64 THEN GOTO60
40 PRINT D$
50 GOTO 20
60 PRINT D$
70 CLOSE 1
```

Mit diesem Programm können Sie Sätze oder ähnliches einlesen, also sehr wichtig bei einer Textverarbeitung oder dergleichen Die Funktion des Programmes ist die gleiche, wie es bei der GET-Anweisung beschrieben wurde.

5.10 Steuerung der Datasette per Programm

Bei dieser Gelegenheit möchten wir nicht vergessen, daß Sie natürlich auch die Möglichkeit haben, während des normalen Programmablaufes den Datenrecorder anlaufen zu lassen. Dies kann ganz besonders dann wichtig sein, wenn Sie zu Beispiel ein Programm geschrieben haben, und im Anschluß daran ein neues Programm geladen werden soll. Dieses kann nötig sein, wenn Sie zu Ihrem Programm eine Einleitung machen wollen und in Anschluß daran das Hauptprogramm geladen werden soll. Das folgende Programm prüft zunächst, ob eine Bandtaste gedrückt ist.

(Achtung: der VC-20 kann zwar prüfen, ob eine Taste gedrückt wurde, aber nicht, welche Taste)

```
10 A=9*4096+256+15+16
20 IF PEEK (A) AND 64 THEN 50
30 PRINT "BITTE BANDTASTE DRÜCKEN"
40 IF PEEK(A) AND 64 = 0 THEN 40
50 X
```

Für das in Zeile 50 stehende X geben Sie bitte einen der folgenden POKES ein, je nachdem, welchen Sie benötigen.

Einschalten des Recorders : POKE 37148,252

Ausschalten des Recorders : POKE 37148,254

So nutzen Sie den Joystick in Ihren Programmen

Der Joystick wird beim VC-20 an einem neunpoligen Sockel angeschlossen.

```
  1   2   3   4   5

  O   O   O   O   O

    O   O   O   O

    6   7   8   9
```

An den Anschlüssen 1-4 sind die Leitungen der zwei Ein/Ausgabe-Bausteine (6522) angeschlossen. Es folgt nun eine Tabelle, aus der Sie die Bedeutung der einzelnen Pinne ersehen können. Bitte schalten Sie Ihren VC-20 aus bevor Sie an dem Anschluß arbeiten, ein Kurzschluß hätte fatale Folgen, denn der VC-20 ist ein hochtechnisches Gerät.

Dies sind die Anschlüsse am Controller-Port und ihre Bedeutung :

PIN	SCHALTERBEZ. JOYS.
1	= JOY 0
2	= JOY 1
3	= JOY 2
4	= JOY 3

5	= LEER
6	= SCHUßKNOPF
7	= LEER
8	= MASSE

Die zwei 6522 VIA-Bausteine haben zwei Ein/Ausgabe Tore. Jedes Tor (Tor A, Tor B) kann unter Zuhilfenahme eines Datenrichtungsregisters als Eingabe- oder Ausgabeport geschaltet werden. Gleich nachdem Sie den Rechner einschalten, sind alle Portleitungen als Eingänge geschaltet. Der Feuerknopf und die Schalter 0,1,2 werden über die VIA #1 gelesen. Der Schalter 3 über die VIA #2.

Die beiden Adressen der Tore sind:

VIA #1 = $9110 HEX = 37136 DEZ.
VIA #2 = $9120 HEX = 37152 DEZ.

Wenn Sie nun die folgenden Zeilen eingeben, werden Sie auf Ihrem Bildschirm den positiven bzw. negativen Wert jeder Joystickstellung angezeigt bekommen.

```
10 DIM JS (2,2) :POKE 37154,0:PA=37137:PB=37152
15 DD=37154
```

```
  20 FORI= OTO2 : FORJ= =TO2: READJS (J,I):NEXTJ,I
  30 DATA -23,-22,-21,-1,0,1,21,22,23
  40 GOSUB 9000 :PRINT JS(X+1,Y+1):GOTO40
9000 POKREDD,127:S3=-((PEEK(PB)AND128)=0):POKEDD,255
9010 P=PEEK(PA):S1=-((PAND8)=0):S2=((PAND16)=0
9015 S0=((PAND4)=0)
9020 FR=-((PAND32)=0):X=S2+S3:Y=S0+S1:RETURN
```

Wenn Sie die Zeilen 20 bis 580 in das obenstehende Programm einfügen, werden Sie jedesmal wenn Sie den Feuerknopf drücken, einen Ton aus Ihrem Fernsehlautsprecher hören.

```
 20 DATA 7,0,1,6,8,2,5,4,3
 30 GOSUB 9000
 40 LET CC=JS (X+1,Y+1)
 90 IF FR=1 THEN GOSUB 500
100 GOTO 30
110 POKE 36877,212
120 FOR L= 15 TO 0 STEP -1
130 POKE 36878,15
140 FOR M=1 TO 10
150 RETURN
```

Nach der Zeile 150 geht es wie im vorherigen Programm weiter. Wenn Sie nun den Feuerknopf drücken und dabei den Regler bewegen, bekommen Sie ein anderes Geräusch. Aber was passiert in dem Programm ? Die Variable FR ist für den Feuerknopf verantwortlich, sie hat immer den Wert 0 . Aber in dem Moment in dem sie gedrückt wird, ändert sich dieser Wert in 1. Bei diesem Programm finden Sie die entsprechende Abfrage in Zeile 90. Die POKES in den unteren Zeilen werden Sie kennen, sie sind für die Lautstärke und den Ton zuständig.

Wenn Ihnen jedoch diese Abfrage zu lang seien sollte, so sollten Sie sich die folgende ansehen. Die Zeilennummerm 100-500 sollten Anweisungen in Ihrem Programm sein. In der Unterroutine wird

direkt auf den schon erwähnten VIA Baustein gesprungen.

```
1000 GOSUB 9000
1010 IF(JS AND4) THEN 100
1020 IF(JS AND16)THEN 200
1030 IF(JS AND 128)THEN 300
1040 IF(JS AND 8) THEN 400
1050 IF(JS AND32) THEN 500
1060 GOTO 1000
:
:
9000 POKE 37139,0:POKE 37154,127
9010 J1=PEEK (37137)
9020 J2=PEEK (37152)
9030 JS=J1ORJ2:RETURN
```

An der Stelle der zwei Doppelpunkte muß nun Ihr Programm stehen. Wobei Sie sich natürlich nicht an die vorgegebenen Zeilennummern halten müssen. Bei Programmen die auf mehrere Bilder zugreifen, kann es sinnvoll sein die Joystickroutine mehrmals zu wiederholen. Wobei hier natürlich nur der erste Teil von 1000 bis 1060 gemeint ist.

5.11 Abfrage der Paddle-Bewegungen

Auch die Paddels lassen sich mit einem einfachen PEEK-Befehl abfragen. Zwei Speicherstellen des VC-20 enthalten jeweils den digitalisierten Wert der waagerechten bzw. senkrechten Richtung:

PEEK(36872) Wert der horizontalen Richtung von Obis 255
PEEK(36873) Wert der vertikalen Richtung von Obis 255

5.12 Die Programmierung der Funktionstasten

Der VC-20 hat als einer der ersten Heimcomputern die aus der kommerziellen Datenverarbeitung bekannten sehr nützlichen Funktionstasten. Die übersichtlich angeordneten Tasten mit den Kennzeichnungen F1-F8 haben vom Betriebssystem her beim VC-20 keine bestimmte Aufgabe. Ihren Sinn erhalten sie erst durch entsprechende Anwenderprogramme, die den einzelnen Funktionstasten bestimmte Funktionen zuweisen.

Wenn Sie nun die Funktionstasten für Ihr Programm nutzen wollen benötigen Sie zunächst einmal die ASCII Werte der Tasten. Das folgende Programm zeigt Ihnen den Wert der Taste an, die Sie gerade gedrückt haben.

```
10 PRINT "( CLR/HOME)"
20 GET A$ : IF A$= "" GOTO 20
30 A= ASC (A$)
40 PRINT " DIESE TASTE HAT DEN ASC II-WERT";A
50 GOTO 20
```

Die Werte der einzelnen Funktionstasten lauten:

```
F1 = CHR$ (133)
F2 = CHR$ (137)
F3 = CHR$ (134)
F4 = CHR$ (138)
```

```
F5 = CHR$ (135)
F6 = CHR$ (139)
F7 = CHR$ (136)
F8 = CHR$ (140)
```

So, wie Sie mit der oben aufgeführten Routine den Wert der einzelnen Tasten bekommen, können Sie diese Werte auch in Ihr Programm einsetzen. Ein Beispiel gibt das folgende Programm:

```
  5 POKE 36879,8
 10 PRINT"(CLR/HOME)"
 20 PRINT"BITTE GEBEN SIE IHREN TEXT EIN"
 30 PRINT"DIE TEXTEINGABE MIT RETURN ABSCHLIESSEN"
 40 INPUT T$
 50 PRINT"(CLR/HOME)"
100 PRINT"ZEIGE TEXT? DRüCKE F1"
110 GET A$:IF A$<> CHR$(133) THEN110
120 IF A$<> CHR$(133) THEN GOTO130
130 PRINT"(CLR/HOME)"
140 PRINT"RVS ON";T$
150 PRINT"NEUEN TEXT EINGEBEN? DRüCKE F3"
160 GET B$:IF B$<> CHR$(134) THEN GOTO150
170 PRINT"EINEN MOMENT BITTE"
180 FORI=1TO3000:NEXT
190 GOTO5
```

Wenn Sie das Programm starten, werden Sie aufgefordert einen Text einzugeben. Das in Zeile 40 stehende T$ ist die Variable für den eingegebenen Text, der in der Zeile 140 wieder ausgegeben wird. In den Zeilen 110 und 160 werden die zwei Funktionstasten abgefragt. Erst wenn eine der Tasten gedrückt wird geht das Programm zur nächsten Anweisung . Dieses ist der große Vorteil der "funktionslosen" Funktionstasten, denn Sie müssen sich die Vorteile dieser nicht Tasten durch die Aufgabe anderer erkaufen. Ein weiter Vorteil ist die übersichtliche Anordnung der Funktionstasten in einem Block. Somit können die Funktionstasten wesentlich zum Bedienungskomfort eines Programmes beitragen.

Noch ein Tip zur Verwendung der Funktionstasten : besonders bei komplexen Programmen empfiehlt es sich, stets bestimmte Funktionstasten mit den selben Befehlen zu belegen, zum Beispiel:

F3 = Programmende
F4 = Eingabe berichtigen
F7 = Daten abspeichern

So werden Ihre Programme leichter bedienbar. Ein fremder Anwender findet sich dann auch ohne blättern in der Bedienungsanleitung gut und schnell zurecht.

5.13 Wie man Programme vom VC-20 auf CBM-Rechner überträgt

Es ist hinreichend bekannt, daß VC-20 Kassetten von anderen Commodore-Rechnern nicht oder nicht richtig gelesen werden können. Hier soll eine Möglichkeit aufgezeigt werden, dies ohne großen Aufwand oder Programmierung in Maschinensprache zu tun.

Für alle, denen die Speicheraufteilung des VC-20 nicht geläufig ist, hier eine kurze Zusammenfassung: Der Basic-Bereich beginnt bei dem Grundgerät bei $1000 und reicht bis $1E00, mit 3K-RAM von $0400 bis $1E00, bei mehr als 3K-RAM von $1200 bis $4000, oder $6000 bis $8000, je nach Größe der Erweiterung.

Grundvoraussetzung für eine erfolgreiche übertragung ist, daß das Programm nicht in Maschinensprache-Format auf Kassette gespeichert ist.

Bei diesem Format handelt es sich um ein speziell für den VC-20 erfundenes Kassettenformat, das eine Verschiebung des Programmes beim Laden verhindert. In diesem Format gespeicherte Programme müssen zuerst auf dem VC-20 mit LOAD geladen werden und anschliessend mit SAVE wieder auf Band geschrieben werden. Außerdem muß der CBM-Rechner mindestens 4K-Ram mehr haben, als für das Programm wirklich benötigt wird.

Laden Sie nun das Programm auf dem CBM-Rechner und geben Sie in Zeile 1 die folgenden Befehle ein und starten das Programm mit "RUN" :

```
A=PEEK(251):B=PEEK(252):POKE1025,A:POKE1026,B:POKE1027,1:POKE1028,0
```

Löschen Sie jetzt die Zeile 1

Das Programm befindet sich nun im Speicher und kann geändert oder aber ausgeführt werden.

5.14 Programme, die sich selbst starten

Beim VC-20 gibt es die Möglichkeit, Programme direkt nach dem Einschalten automatisch zu starten. Solche Programme (Maschinenpro-gramme) müssen im ROM oder EPROM Adressbereich von $A000 bis $BFFF liegen. Damit der VC-20 das Autostartprogramm erkennt, müssen die ersten 9 BYTES folgendes enthalten:

$A000 STARTADRESSE DES PROGRAMMS, LOW BYTE
$A001 STARTADRESSE DES PROGRAMMS, HIGH BYTE
$A002 NMI-VEKTOR (RESTORE), LOW BYTE
$A003 NMI-VEKTOR, HIGH BYTE
$A004 - $A008 $41, $30, $C3, $C2, $C2, $CD TEXT "A0 CBM"

5.15 Programmierung des USER PORT

Die acht Datenleitungen des USER-PORT können einzeln auf Ein- oder Ausgabe programmiert werden. Dazu werden die entsprechenden BITS des Datenrichtungsregisters gesetzt (=Ausgang) oder gelöscht (=Eingang). Die Adresse des Datenrichtungregister ist 37138. Die eigntlichen Daten werden in das Datenregister Adresse 37136 geschrieben oder von dort gelesen.

Belegung an der Unterseite des USER-PORTS:

N und A= GND (Masse), B= CB1, M= CB2, C bis L= PB0 bis PB7.

5.16 Ein Tip zu den Commodore RS-232 Schnittstellen

Bei den Commodore VC-20 Schnittstellen VC-1011 RS 232 TTY mit 20 mA Stromschleife und VC-1011 RS 232 Terminal kann es vorkommen, daß Daten verlorengehen. Mit dieser Abfrage vor der Ausgabe eines BYTES ist es möglich, diesen Fehler zu korrigieren:

```
100 IF (PEEK(37136)AND192) <> 192 THEN 100
```

5.17 So nutzen Sie Ihre Disketten doppelt

Wie Sie sicher wissen, sind preiswerte Dissketten in der Regel nur einseitig beschreibbar.

Wenn Sie sich Ihre Diskette einmal genau ansehen, werden Sie bemerken, daß sich auf der rechten Seite der Diskette eine kleine Einbuchtung befindet. Diese Einbuchtung dient der Floppy dazu, den Unterschied zwischen einer geschützten, also nicht mehr beschreibbaren oder einer beschreibbaren Diskette zu erkennen. Drehen Sie Ihre Diskette um und stecken Sie so in das Laufwerk, so findet das Laufwerk keine Einbuchtung und geht davon aus, daß die Rückseite der Diskette nicht beschreibbar ist. Da bei den meisten Diskettenfabrikaten aber auch die Rückseite der einfachen Disketten physisch beschreibbar ist, brauchen Sie nur für die fehlende Einbuchtung zu sorgen.

Nehmen Sie nun eine andere Diskette zur Hand, legen sie mit der Einbuchtung nach links auf die andere und zeichnen mit einem Bleistift vorsichtig die Einbuchtung nach. Jetzt brauchen Sie nur noch mit einer Schere die eingezeichnete Einbuchtung auszuschneiden. Der Erfolg dieser vielleicht etwas mühsamen Ausschneidearbeit wird sich sofort auszahlen, denn sie haben nun die doppelte Speicherkapazität auf einer Diskette. Am besten rüsten Sie Ihre Disketten vor dem ersten Benutzen um. So vermeiden Sie einen Datenverlust durch diese Aktion. Bitte gehen Sie in jedem Fall sorgfältig vor und bedenken Sie, daß kein Hersteller oder Händler auf entsprechend präparierte Disketten mehr Garantie gibt.

5.18 Basic-Programme mit jeder Erweiterung

Wer ist nicht erbost, wenn er gerade eine Speichererweiterung gekauft hat und zu Hause feststellt, daß seine eigenen Programme nicht mehr laufen. Für Ihre eigenen Programme können die folgenden Zeilen sehr hilfreich sein.

```
BILDSCHIRM   : BS=4*(PEEK(36866)AND128)+64*(PEEK(36869)AND120)

FARBE        : FA=4*(PEEK(36866)AND128+37888
```

Sie müssen also diese zwei Zeilen in Ihre Programmen einfügen. Dieses geschieht am besten am Anfang des Programmes. BS gibt die erste Position im Bildschirm-RAM , FA die erste Position im Farb-RAM an. Somit können Sie Ihre Programme auf jeder Erweiterung laufen lassen. Bitte vergessen Sie nicht, daß Sie diese Zeilen natürlich nicht in Maschinenspracheprogrammen anwenden können. Alles was Sie nun noch machen müssen ist, für jedes Bildschirm-Poke nur POKE BS,X und für jeden Bildschirm-Poke nur POKE BS,X eingeben. Das "X" steht für den jeweils dazugehörigen Wert der dem Poke folgen soll. Wenn Sie also einen Farb-POKE haben, der den Bildschirm auf schwarz schaltet, müssen Sie nur POKE FA,8 eingeben. Was macht man aber, wenn man ein Maschienenspracheprogramm hat für eine 3K Erweiterung besitzt, aber nur eine 16K oder mehr Erweiterung hat. Das folgende kleine Programm macht Ihrem VC-20 vor nur eine 3K Erweiterung angeschlossen zu haben. Es ist vollkommen egal ob es nun 16 oder 32K sind. Sie müssen nur vor dem laden des Programms die nun folgenden Zeilen eingeben.

```
POKE 641,0
POKE 642,4
POKE 643,0
POKE 644,30
POKE 648,30
SYS 64824
```

Nun wird auf Ihrem Bildschirm die normale Anzeige einer 3K Erweiterung erscheinen und Sie können nun Ihr Programm wie gewohnt einladen. Wollen Sie nun nach dem spielen wieder über den vollen Speicherplatz verfügen müssen Sie nur SYS 64802 eingeben. Nach Eingabe dieses Befehles können sie nun wieder wie gewohnt mit Ihren VC-20 arbeiten.

5.19 Programme retten bei OUT OF MEMORY ERROR

Sollten Sie grössere Programme schreiben so werden Sie unweigerlich an die Leistungsgrenzen des VC-20 stoßen. Ein Zeichen dafür ist, wenn auf den Bildschirm "Out of Memory" erscheint. In solchen Fällen heißt es,Ruhe bewahren. Speichern Sie das Programm ab, ohne ihm einen Namen zu geben. Geben Sie statt "SAVE" nur "S Shift A " ein, somit haben Sie wenigstens Ihr schwer erarbeitetes Programm gerettet. Sie können es dann mit einer Speichererweiterung wieder einladen. Viel Speicherplatz wird auch dadurch verschenkt, daß zwischen den einzelnen Befehlen immer eine Leerstelle gelassen wird. Sie können ihre Programmschritte also immer ohne Leertaste eingeben. Außerdem ist es für den Programmierer wichtig, auch während des Schreibens über den noch freien Speicherplatz im Bilde zu sein. Geben Sie dazu nur die folgende Zeile ein: PRINT FRE (0) und nach Drücken der RETURN Taste werden Sie über den noch freien Speicherplatz informiert.

5.20 Der VC-20 als (scheinbarer) Speicher-Riese

Sie haben natürlich auch die Möglichkeit,Ihren VC-20 "optisch" zu erweitern. Geben Sie bitte die folgenden Zeilen ein und sehen, was passiert : POKE 56,255 danach Return,in die nächste Zeile geben Sie bitte folgendes ein:

SYS 58238

Na, was sagen Sie dazu! Ist das nicht überraschend, was nun auf dem Bildschirm erscheint. Fragen Sie nun noch den freien Speicherplatz ab und Sie werden sehen, daß durch die Eingabe zweier Befehle der VC-20 "ganz in Ihrer Hand" ist.

Probieren Sie noch ein bisschen mit anderen Pokes unter 100 oder darüber. Sie können Ihr Ergebnis auch gleich durch "? fre (0)" abfragen. Viel Spaß !!

5.21 Schiebung!! oder wenn der Bildschirm schief steht

Bei verschieden Graphikprogrammen für den VC-20, die aus den USA importiert werden, erleben deutsche VC-20 Besitzer manchmal eine kleine Enttäuschung: das Bild steht nicht in der Mitte, sondern in der linken oberen Ecke.

Weniger bekannt ist die Möglichkeit, hier Abhilfe zu schaffen: Commodore hat neben anderen Dingen auch einen Video-Controller in den VC-20 eingebaut. Diesen mysteriösen Chip können wir hier für unsere Zwecke dienstbar machen. Die Basisadresse des Videocontrollers ist $9000. Dies ist auch gleich die Adresse für die horizontale Position des Bildes auf dem Bildschirm. In dezimal lautet diese Adresse 36864. Ein Register weiter oben, nämlich in 36865, befindet sich die vertikale Position des Bildes. Zwei einfache POKE-Befehle können also das Bild an jede Stelle auf dem Bildschirm verschieben. Die richtigen Inhalte für diese Register können Sie durch PEEK-Befehle aus den beiden Positionen erhalten, zweckmäßigerweise direkt nach dem Einschalten. In den folgenden Beispielen wird der Bildschirm einmal nach rechts oben und nach links oben verschoben.

Nach rechts oben:

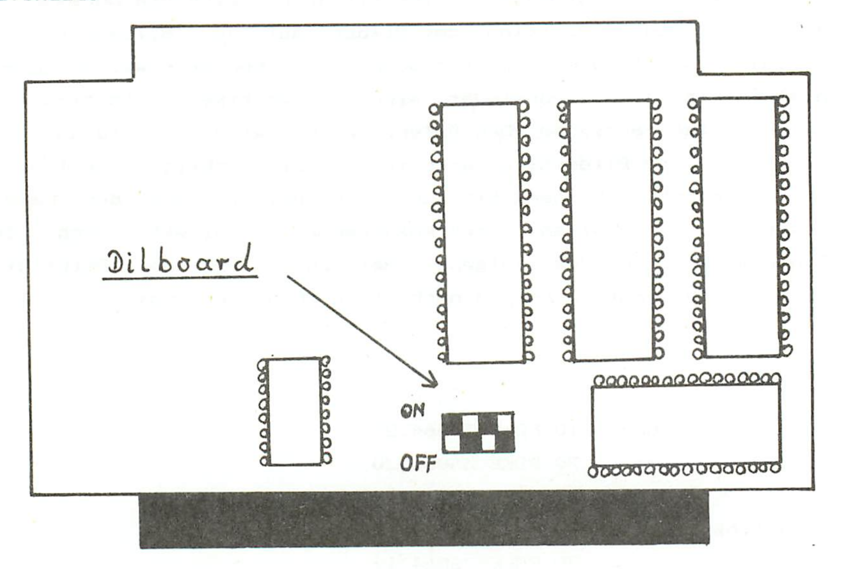
```
10 POKE 36864,20
20 POKE 36865,20
```

Nach links oben

```
10 POKE 36864,10
20 POKE 36864,12
```

5.22 Veränderung des Speicherbereichs der 8-Ram Erweiterung

Die 8-k Erweiterung des VC-20 bietet den großen Vorteil, daß man die Möglichkeit hat, verschiedene Speicherbereiche einzustellen. Wenn Sie die 8-k Erweiterung zur Hand nehmen, so werden Sie feststellen, daß sie auf der Rückseite eine Kreuzschlitzschraube besitzt. Wenn Sie nun diese herausschrauben, können Sie die Erweiterung öffnen. Sie müssen jedoch beachten, daß Sie mit einem Schraubenzieher an den oberen Schlitzen das Gehäuse aufhebeln. Wenn Sie nun die Erweiterung betrachten,werden Sie bemerken,daß sich auf ihr ein Dilboard mit 4 Schaltern befindet.



Wenn die Erweiterung aus dem Werk kommt,ist sie auf folgenden Bereich eingestellt: HEX 2000-3FFF
DEZ 8192-16383

Diese Einstellung ist so für den ausschließlichen Gebrauch der 8k-Erweiterung gedacht, also ist der 4. Schalter an.

Wenn Sie zwei 8k-Erweiterungen benutzen, müssen Sie die Schalter

wie folgt einstellen: alle Schalter, bis auf Nr.3 auf aus. Die Erweiterung liegt nach dem Umstellen dann in den folgenden Bereichen: HEX 4000-5FFF

DEZ 16384-24575

Wenn Sie eine 8k Ram Erweiterung zusammen mit einer 16k Ram Erweiterung benutzen, müssen Sie alle Schalter, bis auf die Nr.2 auf aus stellen. Die Erweiterung liegt dann in den folgenden Bereichen: HEX 6000-7FFF

DEZ 24576-32767

Beim Abspeichern von Maschinenprogrammen ist es sinnvoll, sie in einen Bereich zu legen, der vom Basic nicht berührt wird. So liegen alle Spielmodule von Commodore in diesem Bereich. Der VC-20 zeigt dann die 8K Ram nicht mehr als Erweiterung des BASIC-Speicherplatzes auf dem Bildschirm, doch das sollte Sie nicht weiter beunruhigen. Um die Erweiterung in den Maschinensprache-Bereich zu legen, müssen Sie alle Schalter, bis auf die Nr.1,auf aus stellen. Die Erweiterung liegt dann in den folgenden Bereichen: HEX A000-BFFF

DEZ 40960-49151

Sie haben auch bei der 16K Erweiterung die Möglichkeit, ein Dilboard einzubauen,um sie in andere Bereiche zu legen. Bei der 16K Erweiterung ändern sich die Adressen wie folgt : Anfang Bildschirm von 7680 auf 4096 und der Anfang des Farb-Ram von 38400 auf 37888. Dieses ist bitte zu beachten. Wenn Sie nun zwei 16K-Erweiterungen einstecken, werden Ihnen die zusätzlichen 16k nicht angezeigt, Sie können sie aber für Ihre Programme, die Sie in Maschinensprache, geschrieben haben nutzen, aber nur unter der Vorausetzung, daß Sie die oben genannten Dilschalter auch eingesetzt haben.

5.23 Disk Menü

Wer kennt nicht das Problem, das sich ergibt wenn man mit der Diskettenstation arbeitet. Es fängt an beim Formatieren der Disketten und hört auf beim Säubern der Disketten. Das folgende Programm nun soll Ihnen helfen besser und leichter mit Ihrer Diskettenstation unzugehen. Wenn Sie das Programm eingegeben haben speichern Sie es auf einer neuen formatierten Diskette ab. Das Programm ist so aufgebaut, daß Sie sobald Sie mit einer neuen Diskette arbeiten erst das Menü enladen und sich dann über das Menü den Inhalt der Diskette ansehen können. Sie brauchen beim Umgang mit den Programm nichts zu beachten. Sie wählen nur unter der entsprechenden Nummer nur den von gewünschten Teil ein und das Programm führt ihn aus.

Doch nun zur Programmerklärung. Sobald Sie das Porgramm mit RUN gestartet haben, erscheint eine Maske auf dem Bildschirm (Zeile 30 bis 160). In der Zeile 190 ist eine INPUT Anweisung, je nach Eingabe die Sie machen wählt das Programm einen anderen Programmteil an. Dieses können Sie in der Zeile 200 nachlesen. Wenn Sie sich nun die einzelen Unterroutinen ansehen, werden sie bemerken das hier nur die Ihnen schon bekannten Diskettenbefehle eingebunden sind. Bitte geben Sie das Programm sehr gewissenhaft ein, den nur so können Sie einen fehlerfreien Ablauf erreichen. Es wurden absichtlich keine Frabpokes gesetzt, denn jeder Programmierer hat da seine eigenen Vorstellungen. Diese Farbpokes werden in die Zeile 10 gesetzt. Doch nun folgt das Listing des hier besprochenen Programmnes.

```
10 print"■":rem: hier koennen sie die farbpokes setzen
20 print"■     Disk   Menue      "
30 print"---------------------"
40 print
50 print"1=Disk Directory     "
55 print"2=drucke Directory   "
60 print"3=formatiere     Disk"
70 print"4=initialisiere Disk"
80 print"5=copiere         File"
90 print"6=umbenennen File    "
100 print"7=loesche    File    "
110 print"8=Diskette saeubern"
120 print"9=schreibe    Menue  "
130 print"10=Disk Status       "
135 print"11=lade Programm     "
140 print"12=  ■Ende■          "
150 print
160 print"--------------------":print
170 print"■Bitte waehlen■";
190 input choise
200 on choise gosub 250,800,300,350,400,450,500,550,600,650,900,700
210 goto10
250 print"■"
251 open1,8,0,"$"
252 get #1,a$,b$
254 get #1,a$,b$
256 get #1,a$,b$
258 c=0:ifa$<>""then c=asc(a$)
260 ifb$<>""then c=c+asc(b$)*256
262 print"■ "mid$(str$(c),2);tab(1);" ";
264 get #1,b$:ifst<>0then282
266 ifb$<>chr$(34)then264
268 get #1,b$:if b$<>chr$(34) then printb$;:goto268
270 get #1,b$:if b$=chr$(32) then 270
272 printtab(10);:c$=""
274 c$=c$+b$:get#1,b$:if b$<>""then274
276 printtab(18)"■ "left$(c$,3)
280 if st=0 then 254
282 print"■Blocks frei "
284 close 1:print:print:print"druecke ■return■ fuer Menue";:inputx$:return
300 print"■":print:print
```

```
305 print"Diskette fuers formatieren einlegen":print
310 print"Eingabe des Disketten-Namen          ";:input disk$:print
320 print"Eingabe der Disketten-Nr.            ";:input ext$
325 macro$="n:"+disk$+","+ext$
330 open 15,8,15,macro$
340 close 15:macro$="":return
350 print"":print:print
360 print"Diskette einlegen zum initialisieren":print
370 print"druecke return fuer Initialisierung";:inputx$
380 open 15,8,15,"i"
390 close 15:return
400 print"":print:print
410 print"copiere File (Name)          ";:input disk$:print
420 print"neuer   File  Name           ";:input nws$
425 macro$="c:"+nws$+"="+disk$
430 open 15,8,15,macro$
440 close 15:macro$="":return
450 print"":print:print
460 print"Eingabe des alten Filenamen";:input disk$:print
470 print"Eingabe des neuen Filenamen";:input nws$
475 macro$="r:"+nws$+"="+disk$
480 open 15,8,15,macro$
490 close 15:macro$="":return
500 print"":print:print
510 printtab(2)"loesche File (Namen)";:input disk$:print
511 fori=1to5:print:next:rem:poke fuer farbe
512 printtab(2)"                                    "
513 printtab(2)"                                    "
514 printtab(2)"                                    "
515 printtab(2)"                                    "
516 printtab(2)"                                    "
517 printtab(2)"                                    ":print
518 print""
519 goto1000
520 printtab(2)"druecke Return       ";:input x$
530 macro$="s:"+disk$
535 open 15,8,15,macro$
540 close 15:macro$="":return
550 print"":print:print
560 print"*** Achtung *** offene Files werden geloescht":print:print
570 print"druecke (return) fuer Disketten-Saeuberung      ";:input x$
580 open 1,8,15,"v"
590 close 1:return
600 print"":print:print
```

```
610 print"■schreibe neues Menue■":print
620 print"■druecke Return       ■";:input x$
625 open 1,8,15
630 save "Menue",8
635 close 1
640 return
650 open 1,8,15
660 input#1,a,b$,c,d
670 print"■":print:print
680 print"■Fehler Status■":print:print"■Fehler # ■";a
681 print"■Spur           ■";c,"■Sector      ■";d
685 print:print"■0 = kein Fehler"
690 close 1:print:print:print"druecke ■return■ fuer Menue";:inputx$:return
700 print"■":fori=1to10:print:next
710 printtab(3)"■*******    A c h t u n g    ********■
720 print
725 printtab(3)"■    M e n u e          E n d e       ■
730 end
800 open4,4,7
810 print"■":fori=1to10:print:next
850 print"■Druckausgabe Directory■"
851 open1,8,0,"$"
852 get #1,a$,b$
854 get #1,a$,b$
856 get #1,a$,b$
858 c=0:ifa$<>""then c=asc(a$)
860 ifb$<>""then c=c+asc(b$)*256
861 print#4,chr$(27);"d";chr$(10);chr$(0);
862 print#4,mid$(str$(c),2);chr$(9);"  ";
864 get #1,b$:ifst<>0then882
866 ifb$<>chr$(34)then864
867 print#4,chr$(27);"d";chr$(25);chr$(0);
868 get #1,b$:if b$<>chr$(34) then print#4,b$;:goto868
870 get #1,b$:if b$=chr$(32) then 870
871 print#4,chr$(27);"d";chr$(18);chr$(0);
872 print#4,chr$(9);:c$=""
874 c$=c$+b$:get#1,b$:if b$<>""then874
875 print#4,chr$(27);"d";chr$(35);chr$(0);
876 print#4,chr$(9);left$(c$,3)
880 if st=0 then 854
881 print#4,chr$(27);"d";chr$(10);chr$(0);
882 print#4," Blocks frei":close 4
884 close 1:print:print:print"druecke ■return■ fuer Menue";:inputx$:return
900 qo$=chr$(34)
```

```
910 input"welches Programm";pro$
920 print"load";qo$;pro$;qo$;",8"
930 print"run"
990 fori=1to6:poke622+i,13:next:poke156,6
999 end
1000 printtab(2)"Korrektur";:
1010 inputko$
1015 ifko$<>"j"andko$<>"n"thenprint"":goto1000
1020 ifko$="j"thengoto500
1030 ifko$="n"thengoto520
1040 end
ready.
```

5.24 Der Trick mit dem LIST

Wen ärgert es nicht, daß man bei der Korrektur von Programmen ein 'sehr schnelles Auge' braucht um die Änderungen auch schnell und sicher durchführen zu können. Denn Sie werden schon gemerkt haben, daß beim LIST-Befehl die einzelnen Zeilen recht schnell durchlaufen, so daß es durchaus pasieren kann, daß man ein paar Fehler innerhalb des Programmes übersieht.

Auch die Verwendung der CTRL-Taste während des LIST hat nur eine schwache Verzögerung zur Folge, die für eine genaue Betrachtung der Zeile nicht ausreicht.

Es gibt aber einen Trick mit dem man den Ablauf des LIST Befehls sehr verlangsamen, ja sogar anhalten kann. Hierzu dient eine bestimmte Speicherstelle des VC-20, die nur selten dokumentiert wird. Es handelt sich hierbei um die Adresse 37879. Durch Veränderung des Wertes in dieser Adresse kann die Uhr des VC-20 schneller oder langsamer gemacht werden. So wird z.B. durch ein POKE 37879,0 die Uhr ca. 60 mal schneller als normal. Das hat zur Folge, daß beim LIST der Durchlauf sehr verlangsamt wird. Durch drücken der SHIFT-Taste wird jede Zeile recht gut lesbar. Drückt man dann noch die CTRL-Taste, so wird der Durchlauf ganz angehalten. Ansonsten läßt sich die Änderung jeder Zeile bzw. der Abbruch durch RUN/STOP wie gehabt durchführen.

Ein Effekt dieses POKEs ist außerdem das schnellere Blinken des Cursors und die schnellere Bewegung des Cursors über den Bildschirm. Wird der SUPER EXPANDER verwendet, so werden auch verschiedene Grafik-Befehle langsamer ausgeführt.

5.25 Unnew

Haben Sie auch schon einmal aus Versehen ein Programm mit "NEW" gelöscht, ohne es vorher abgespeichert zu haben?
Dann ist das folgenden Porgramm genau richtig für Sie.
Das Wesen des "NEW"-Befehls ist es, daß nicht wie vielleicht zu vermuten ist, der gesamte Speicher gelöscht wird, sondern nur die BASIC-Zeiger zurück gesetzt werden. Aus diesem Grund ist es kein größeres Problem, diesen Befehl wieder rückgängig zu machen.
Wichtig ist nur, daß Sie keine neuen Variablen nach dem "NEW"-Befehl benutzt haben.
Das Programm liegt wieder im Kassettenpuffer und wird einfach mit SYS 828 gestartet. Anscchließend ist Ihr BASIC-Programm wieder vorhanden.

```
033C                      .OPT P1
033C                      *=   828
033C A5 2B                LDA  $2B
033E A4 2C                LDY  $2C
0340 85 22                STA  $22
0342 84 23                STY  $23
0344 A0 03                LDY  #3
0346 C8         NULL      INY
0347 B1 22                LDA  ($22),Y
0349 D0 FB                BNE  NULL
034B C8                   INY
034C 98                   TYA
034D 18                   CLC
034E 65 22                ADC  $22
0350 A0 00                LDY  #0
0352 91 2B                STA  ($2B),Y
0354 A5 23                LDA  $22 + 1
0356 69 00                ADC  #0
0358 C8                   INY
0359 91 2B                STA  ($2B),Y
035B 88                   DEY
035C A2 03      T0        LDX  #3
035E E6 22      TDREI0    INC  $22
0360 D0 02                BNE  * + 4
0362 E6 23                INC  $22 +1
0364 B1 22                LDA  ($22),Y
0366 D0 F4                BNE  T0
0368 CA                   DEX
0369 D0 F3                BNE  TDREI0
036B A5 22                LDA  $22
036D 69 02                ADC  #2
036F 85 2D                STA  $2D
0371 A5 23                LDA  $22 + 1
0373 69 00                ADC  #0
0375 85 2E                STA  $2E
0377 20 63 A6             JSR  $A663
037A 4C 67 E4             JMP  $E467
```

KAPITEL 6 - BASIC-ERWEITERUNGEN UND TOKENS

Um Ihren VC-20 richtig zu verstehen, müssen Sie sich natürlich auch mit dem Aufbau der verschiedenen Routinen, die für das eigentliche interpretiern zuständig sein, auskennen. Dazu ist es wichtig zu wissen, in welcher Form das BASIC Programm abgespeichert wird, und wie es modifiziert werden kann. Der erste Schritt dazu, ist die Kenntnis der sogenannten TOKEN.

6.1 TOKENS - Was ist das ?

Wie Sie wissen werden, kann jeder Computer, auch Ihr VC-20, nur mit Zahlen umgehen. Genauer gesagt mit 2 Zahlen: 0 und 1. Diese elektrische Zustände bilden zusammen eine Information. Geben Sie also den Befehl PRINT ein, so kann der Computer diese Buchstabenkombination überhaupt nicht verstehen. Er muß diesen Befehl in eine Zahl umwandeln. Anderes ist das mit dem Teil nach PRINT. Steht dort zum Beispiel irgendein Satz, der also auf dem Bildschirm ausgegeben werden soll, so wandelt der Computer nicht den ganzen Satz in eine Zahl um, sondern Zeichen für Zeichen. Im Speicher wird also für den Befehl PRINT genauso viel Platz benötigt, wie für ein einfaches 'A'. Die Zahl, die aus dem Befehl PRINT errechnet wurde nennt man TOKEN. Jeder BASIC-Befehl, jede Funktion und jedes Sonderzeichen, ist so als Zahl zu kodieren. Um nun später den Aufbau einer BASIC-Zeile zu verstehen, müssen wir uns an dieser Stelle mit den TOKEN beschäftigen.

Auf der nächsten Seite nun, folgt eine Tabelle sämtlicher TOKEN. Auf ihre Anwendung kommen wir später noch.

Liste aller TOKEN

128	END	160	CLOSE	192	TAN
129	FOR	161	GET	193	ATN
130	NEXT	162	NEW	194	PEEK

131	DATA	163	TAB(	195	LEN
132	INPUT#	164	TO	196	STR$
133	INPUT	165	FN	197	VAL
134	DIM	166	SPC(	198	ASC
135	READ	167	THEN	199	CHR$
136	LET	168	NOT	200	LEFT$
137	GOTO	169	STEP	201	RIGHT$
138	RUN	170	+	202	MID$
139	IF	171	-	203 - 254	unbenutzt
140	RESTORE	172	*	255	(PI)
141	GOSUB	173	/		
142	RETURN	174			
143	REM	175	AND		
144	STOP	176	OR		
145	ON	177	>		
146	WAIT	178	=		
147	LOAD	179	<		
148	SAVE	180	SGN		
149	VERIFY	181	INT		
150	DEF	182	ABS		
151	POKE	183	USR		
152	PRINT#	184	FRE		
153	PRINT	185	POS		
154	CONT	186	SQR		
155	LIST	187	RND		
156	CLR	188	LOG		
157	CMD	189	EXP		
158	SYS	190	COS		
159	OPEN	191	SIN		

An Hand dieser Tabelle, kann man nun leicht jedes beleibe BASIC-Programm analysieren. Dazu ist es allerdings notwendig, etwas in Maschinensprache zu arbeiten. Sie können ein BASIC Programm auf zwei verschiedene Art und Weise analysieren. Die einfachste Möglichkeit ist die Anschaffung eines sogenannten Monitors. Mit diesem Monitor können Sie nun leicht jede Adresse des VC-20 ausleisten, seine Register überwachen, Programme laden und

Speichern, Programme verändern und vieles mehr. Die zweite Möglichkeit wäre, mit der PEEK-Funktion Wert für Wert aus dem Speicher zu lesen und auf Bildschirm oder Drucker auszugeben. Danach könnten mögliche Änderungen mit dem POKE-Befehl durchgeführt werden. Aber Vorsicht! Ein BASIC-Programm kann durch solche Änderungen, wenn diese nicht korrekt sind, auf leichte Art und Weise zerschgossen werden.

Sehen wir uns zunächst einmal den Aufbau einer BASIC-Zeile an. Um an die Adresse der ersten BASIC-Zeile zu gelangen, müssen Sie die Werte in den Adressen 4097 und 4098 auswerten.

```
X=PEEK(4097)+256*PEEK(4098)
```

Ein Ausgabe mit dem Monitor könnte folgendermaßen aussehen:

```
.:   1000 00 10 10 0A 00 99 22 56
.:   1008 43 2D 32 30 22 3A 80 00
.:   1010 00 00 xx xx xx xx xx xx
```

Was bedeuten nun die einzelnen Speicherplätze ? Die erste Adresse ist jeweils der Speicherplatz, an der die einzelnen Werte stehen. Da es sich hier um ein BASIC-Programm handelt, ist diese Adresse $1000 oder dezimal 4096. Danach steht die eigentliche Verbindungsadresse zwischen erster und nächster Programmzeile. In unserem Beispiel bedeutet das, daß die nächste Programmzeile bei der Adresse $1010 oder dezimal 4112 beginnt. Die nächste Adresse stellt die verwendete Zeilennumer dar. Hier lautet sie $0A oder dezimal 10. Danach folgen die oben besprochenen TOKENS. Die Zeile endet mit dem Wert 0. Nach diesem Zeilenende könnte das Programm mit einer neuen Zeilennummer normal weitergehen. In unserem Beispiel steht hier die Adresse 0. Diese Adresse sagt dem BASIC-Interpreter, der die ganze Zeile abarbeitet, daß hier das Programm beendet ist.

In BASIC würde das oben angegebene Beispiel so aussehen:

Der reine Anwender wird mit diesen Informationen vielleicht noch nicht so viel anfangen können. Aber derjenige, der seinen Computer von "Anfang bis Ende" kennen will, der wird rasch die Möglichkeiten entdecken, die mit diesen Kenntnissen aufgedeckt werden.

Auf den nächsten Seiten finden Sie nun noch eine Reihe erstklassiger BASIC-Erweiterungen - also Maschinenprogramme, die dem Programmierer sehr nützlich sein können. Lassen Sie sich von diesen Ideen inspirieren und entwickeln Sie selbst einmal Maschinenprogramme, oder beschäftigen Sie sich an einem regnerischen Tag doch einmal ausführlich mit dem BASIC des VC-20. Sie werden sehen, wie schnell auch Sie zu einem VC-20 Experten werden.

6.2 Append - BASIC-Programme werden verbunden

Sicher haben Sie auch schon das Problem gehabt, daß Sie zwei BASIC-Programme aneinanderfügen wollten. Vielleicht haben Sie den Trick mit dem Bildschirmeditor benutzt (Zeilen auf dem Bildschirm stehen lassen, zweites Programm laden und die Zeilen auf dem Bildschirm wieder einlesen). Dieses Verfahren hat aber den besonderen Nachteil, daß die Anzahl der Zeilen auf die Kapazität des Bildschirms beschränkt ist. Wir wollen Ihnen nun ein Verfahren zeigen, mit dem Sie beliebig lange Programme verbinden können.
Bei diesem Verfahren wird ein reines Anhängen durchgeführt. Dies bedeutet, daß das nachgeladene Programm größere Zeilennummern haben muß als das erste Programm. Sie können sich also nun eine Unterprogrammbibliothek aufbauen. Dabei sollten die Unterprogramme möglichst große Zeilennummern haben. Doch nun genug der Vorrede, wie fügen Sie nun BASIC-Programme aneinander?
Tippen Sie zuerst Ihr nachzuladendes Programm ein und speichern Sie es auf Kassette oder Diskette ab. Nun tippen Sie ihr zweites Programm ein. Nur dieses Programm befindet sich nun in Ihrem Rechner. Tippen Sie nun die folgende Zeile ein:

```
PRINT PEEK(43),PEEK(44)
```

Geben Sie diese Zeile unbedingt im Direktmodus ein. Sie dürfen von jetzt an bis zum Abschluß des Aneinanderfügens keine Änderungen am Programm vornehmen oder eine Variable benutzen! Sie haben mit der obigen Zeile zwei Werte erhalten. Dies sind die Startadressen für Ihr BASIC-Programm. Schreiben Sie sich diese beiden Zahlen auf, sie brauchen Sie gleich noch.
Die Adresse des Ende Ihres BASIC-Programms steht in den Speicherstellen 45 und 46. Der Wert dieser beiden Speicherstellen wird nun in die Speicherstellen 43 und 44 geschrieben. Dies bedeutet praktisch, daß der BASIC-Start an das Ende Ihres alten Programms gelegt wird. Wir machen uns nun zu Nutze, daß der VC-20 BASIC-Programme immer an die Adresse verschiebt, die in den Adressen 43 und 44 steht. Beachten müssen wir noch, daß der

BASIC-Interpreter am Ende der letzten Programmzeile zwei Nullen schreibt um das Ende des Programms zu kennzeichnen. Aus diesem Grund müssen wir die Adresse aus 45 und 46 um zwei vermindern und dann in die Speicherstellen 43 und 46 schreiben. Tippen sie nun ein:

```
POKE43,(PEEK(45)+256*PEEK(46)-2)AND255
POKE44,(PEEK(45)+256*PEEK(46)-2)/256
```

Damit haben Sie nun den BASIC-Start verlegt. Nun können Sie das zweite Programm nachladen:

```
LOAD"name"
```

Wenn Sie nun LIST eintippen, sehen Sie nur das zweite Programm. Tippen Sie nun in die Speicherstellen 43 und 44 die aufgeschriebenen Werte ein:

```
POKE43,...
POKE44,...
```

Nun sind Ihre beiden Programme aneinandergefügt. Wenn Sie LIST eintippen, sehen Sie beide Programme fein säuberlich auf dem Bildschirm.

6.3 AUTO - automatische Zeilennumerierung

Wir wollen Ihnen nun noch ein kleines BASIC-Programm zeigen, daß Ihnen viel Tipparbeit ersparen wird. Das Programm erzeugt automatisch Zeilennummern. Sie geben die Nummer der Startzeile ein und anschließend die Schrittweite der Zeilen.
Von nun an wird nach jedem Drücken der RETURN-Taste eine neue Zeilennummer erzeugt. Das Programm sieht folgendermaßen aus:

```
0 INPUT"START";S:INPUT"SCHRITTWEITE";W:POKE1002,W
1 POKE1000,INT(S/256):POKE1001,S-INT(S/256)*256
2 PRINTS;
3 WAIT198,1:GETA$:PRINTA$;:IFA$<>CHR$(13)THEN3
4 PRINT"RUN5":FORI=631TO634:POKEI,145:NEXT:POKE635,13:POKE636,13
  :POKE198,6:END
5 PRINTCHR$(145);"     ":PRINT:PRINT"        ";CHR$(145);
  CHR$(145);CHR$(145)
6 S=PEEK(1000)*256+PEEK(1001)+PEEK(1002):GOTO1
```

Zeile 0 fragt die Startzeile und die Schrittweite ab. Die Schrittweite wird in der Speicherstelle 1002 sofort zwischengespeichert. Die Startzeile wird in Zeile 1 ebenfalls abgespeichert und ausgedruckt. In Zeile 3 werden nun die Eingaben eingelesen, bis Sie die RETURN-Taste betätigen. In Zeile 4 wird nun zuerst "RUN5" ausgedruckt, dann viermal CURSOR-hoch und zweimal Carriage-Return in den Tastaturpuffer geschrieben. Dadurch wird zuerst die eingetippte Zeile eingelesen und anschließend Zeile 5 gestartet. In dieser Zeile werden nun das "RUN5" und die READY-Meldung gelöscht. Dann wird in Zeile 6 die Variable S mit der neuen Zeilennummer belegt und wieder in Zeile 1 verzweigt, wo diese Zeilennummer wieder abgespeichert und ausgedruckt wird.

6.4 INPUT - Strings >88 Zeichen einlesen

Wenn Sie mit der Floppy arbeiten, wird sicher, besonders im kommerziellen Bereich, eine der Hautpaufgaben das Einlesen von Informationen sein. Diese Informationen sind meistens Strings, wie A$, B$ u.s.w. Das BASIC des VC-20 hat hierfür mehrere Befehle vorgesehen. Das Schreiben eines Strings in eine Datei, egal ob relative oder sequentielle, geschieht mit dem Befehl 'PRINT#' . Ein Beispiel soll dies verdeutlichen:
Sie haben eine Adressenliste. Eine gesamte Adresse steht in einer Stringvariablen, alle Adressen in einem Array, z.B. A$. Nun möchten Sie diese auf Diskette in einem sequentiellen File abspeichern. Gehen wir davon aus, daß Sie 100 Adressen haben. Das folgende kleine BASIC-Programm würde dies übernehmen:

```
10 OPEN 1,8,2,"ADRESSEN,S,W"
20 FOR I=1 TO 100
30 PRINT#1,A$(I)
40 NEXT I
50 CLOSE 1
```

Nach Ausführung dieses Programms sind Ihre Adressen auf Diskette in dem sequentiellen File 'ADRESSEN' gespeichert.
Wir gehen davon aus, daß eine einzelne Adresse ca. 130 Zeichen enthält.
Analog zum 'PRINT#'-Befehl existiert der 'INPUT#'-Befehl. Mit diesem Befehl ist es möglich, Strings wieder einzulesen. Es steht also zu vermuten, daß ein analoges Programm zu dem obigen die Daten wieder einliest. Ein solches Programm könnte nun so aussehen:

```
10 DIM A$(100)
20 OPEN 1,8,2,"ADRESSEN,S,R"
30 FOR I=1 TO 100
40 INPUT#1,A$(I)
50 NEXT I
60 CLOSE1
```

Dieses Programm funktioniert aber leider nur, solange die einzelnen Strings nicht länger als 88 Zeichen sind. Dies ist aber leider nur in den wenigsten Fällen der Fall.
Der Grund für dieses etwas seltsame Verhalten liegt darin, daß die Daten, die von der Diskette kommen, zuerst einmal in einem Zwischenspeicher aufbewahrt werden. Dies wäre ja nicht so schlimm, hätte dieser Speicher nicht die maximale Größe von 88 Zeichen. Erhält der Rechner von der Floppy Strings, die länger als 88 Zeichen sind, so ergibt sich ein 'STRING TOO LONG ERROR'.
Ein weiterer Nachteil des 'INPUT#'-Befehls ist, daß der Doppelpunkt und das Komma als Endezeichen erkannt werden. Somit ist es nicht möglich, diese Zeichen innerhalb eines Strings mitabzuspeichern.
Eine Möglichkeit, diese Nachteile zu umgehen, bietet der 'GET#'-Befehl. Mit diesem Befehl können Sie ein einzelnes Zeichen von der Floppy herunterholen. Ein BASIC-Programm, daß ohne Fehler läuft, wäre nun das Folgende:

```
10 DIM A$(100):I=1
20 OPEN 1,8,2,"ADRESSEN,S,R"
30 GET#1,Z$:IFZ$=13THEN50
40 A$(I)=A$(I) + Z$:GOTO30
50 IF I=100 THEN CLOSE 1 : END
60 I=I + 1 : GOTO 30
```

Dieses Programm liest Ihre Adressen nun wieder richtig ein. Es hat jedoch zwei gewichtige Nachteile. Zum ersten ist es sehr langsam, zum anderen belastet es durch die umfangreichen Stringoperationen sehr die Garbage-collection-Routine. Um dies alles zu umgehen, bleibt uns nichts anderes übrig, als eine kleine Routine zu schreiben, die uns den 'INPUT#'-Befehl ersetzt. Diese kleine Maschinenspracheprogramm gestattet es, einen String beliebiger Länge mit allen Zeichen einzulesen. Aufgerufen wird die Routine mit :

SYS ADRESSE, VARIABLE, LäNGE DES STRINGS, FILENUMMER

- ADRESSE : Dies ist die Startadresse des Programms

- VARIABLE : Dies ist die Stringvariable, in die die Zeichen von der Floppy eingelesen werden sollen.

- LäNGE : Dies ist die Anzahl der Zeichen, die eingelesen werden soll.

- FILENUMMER: Dies ist die Nummer, mit der das File eröffnet wurde.

Bewährt hat sich diese Routine besonders bei relativen Dateien. Sie ist schnell und vergeudet keinen Speicherplatz
Ein Programm, daß diese Routine benutzt, könnte dann so aussehen:

```
10 DIM A$(100)
20 OPEN 1,8,2,"ADRESSEN,S,R"
30 FOR I=1 TO 100
40 SYS 828,Z$,130,1
50 A$(I)=Z$ : GET#1,Z$
60 NEXT I
70 CLOSE 1
```

In diesem Fall wurde angenommen, daß das Programm bei der Adresse 828 liegt, und die Datensätze alle die Länge von 130 Zeichen haben.
Auf der folgenden Seite finden Sie das Assemblerlisting zu dem Programm, assembliert für den Speicherbereich ab 828.

```
90:    033C                          .OPT P1
                          ; INPUT#
                          ;
                          ; AUFRUF MIT SYS ADR,VARIABLE,LAENGE,FILENUMMER
                          ;
200:   033C                CHKCOM    =    $CEFD   ; KOMMA
210:   033C                GETVAR    =    $D08B
220:   033C                FRESTR    =    $D6A3
230:   033C                GETBYT    =    $D79E
240:   033C                STRRES    =    $D47D   ; STRING RESERVIEREN
250:   033C                STRADR    =    $62
252:   033C                SETSTR    =    $D475
255:   033C                DESCRPT   =    $64
260:   033C                TEMP      =    $FB
270:   033C                TEMP2     =    $FC
280:   033C                CHKIN     =    $FFC6
290:   033C                BASIN     =    $FFCF
295:   033C                CLRCH     =    $FFCC
300:   033C                          *=   $033C
310:   033C 20 FD CE                 JSR  CHKCOM
320:   033F 20 8B D0                 JSR  GETVAR  ; VARIABLE HOLEN
325:   0342 48                       PHA
325:   0343 98                       TYA
325:   0344 48                       PHA
330:   0345 20 A3 D6                 JSR  FRESTR
340:   0348 20 FD CE                 JSR  CHKCOM
350:   034B 20 9E D7                 JSR  GETBYT  ; LAENGE IN X-REGISTER
365:   034E 68                       PLA
365:   034F 85 65                    STA  DESCRPT + 1
367:   0351 68                       PLA
367:   0352 85 64                    STA  DESCRPT
368:   0354 8A                       TXA          ; LAENGE IN AKKU
369:   0355 85 FC                    STA  TEMP2
380:   0357 20 75 D4                 JSR  SETSTR  ; PLATZ FUER STRING RESERVIEREN
381:   035A 84 FB                    STY  TEMP
381:   035C A0 00                    LDY  #0
381:   035E 91 64                    STA  (DESCRPT),Y
381:   0360 C8                       INY
381:   0361 8A                       TXA
```

```
381:    0362 91 64              STA  (DESCRPT),Y
381:    0364 C8                 INY
382:    0365 A5 FB              LDA  TEMP
382:    0367 91 64              STA  (DESCRPT),Y
390:    0369 20 FD CE           JSR  CHKCOM
395:    036C A0 01              LDY  #1
395:    036E B1 64              LDA  (DESCRPT),Y
395:    0370 48                 PHA
395:    0371 C8                 INY
395:    0372 B1 64              LDA  (DESCRPT),Y
395:    0374 48                 PHA
400:    0375 20 9E D7           JSR  GETBYT   ; FILENUMMER HOLEN
405:    0378 20 C6 FF           JSR  CHKIN
410:    037B 68                 PLA
410:    037C 85 65              STA  DESCRPT+1
410:    037E 68                 PLA
410:    037F 85 64              STA  DESCRPT
430:    0381 A0 00              LDY  #0
433:    0383 20 CF FF LOOP      JSR  BASIN
440:    0386 91 64              STA  (DESCRPT),Y  ; STRING ERSTELLEN
450:    0388 C8                 INY
450:    0389 C4 FC              CPY  TEMP2
450:    038B D0 F6              BNE  LOOP
460:    038D 4C CC FF           JMP  CLRCH
```

6.5 STRING$

Die Stringverarbeitung im BASIC des VC-20 ist nicht gerade reichhaltig und komfortabel ausgefallen. Aus diesem Grund stellen wir Ihnen in diesem und dem nächsten Abschnitt zwei Programme vor, die dieses Manko etwas lindern. Der erste neue Befehl eröffnet die Möglichkeit, einen String beliebiger Länge zu erzeugen.
Natürlich können Sie dies auch von BASIC mit einer kleinen Routine simulieren, z.B.:

```
10 A$="E"
20 FOR I=1 TO 10 : B$=B$+A$ : NEXT I
```

Das Ergebnis dieses Programms ist ein String, bestehend aus 10 mal dem Buchstaben 'B'. Dieses Verfahren hat jedoch wieder zwei gravierende Nachteile. Wie immer im Vergleich zu Maschinensprache ist dieses Verfahren erheblich zeitaufwendiger. Der zweite Nachteil ist die übergroße Belastung der Garbage-collection. Gerade in dem nicht sehr großen Speicherbereich des VC-20 macht sich diese Routine bei häufiger Stringverarbeitung sehr schnell und oft in längeren Wartezeiten bemerkbar.

Aufgerufen wird die Routine wie folgt :

SYS ADRESSE,VARIABLE,LÄNGE,ASCII-CODE DES ZEICHENS

Ein Beispiel dazu wäre :

SYS 828,B$,10,69

Dieser Aufruf bewirkt das Gleiche wie das vorstehende BASIC-Programm.
Auf der folgenden Seite nun das Assemblerlisting zu dieser Routine.

```
2
90:     033C                    .OPT P1
                         ; STRING$
                         ;
                         ; AUFRUF MIT SYS ADR,VARIABLE,LAENGE,ASCII-CODE DES ZEICHE
NS
                         ;
200:    033C             CHKCOM  =     $CEFD    ; KOMMA
210:    033C             GETVAR  =     $D08B
220:    033C             FRESTR  =     $D6A3
230:    033C             GETBYT  =     $D79E
240:    033C             STRRES  =     $D47D    ; STRING RESERVIEREN
250:    033C             STRADR  =     $62
252:    033C             SETSTR  =     $D475
255:    033C             DESCRPT =     $64
260:    033C             TEMP    =     $FB
300:    033C                     *=    $033C
310:    033C 20 FD CE            JSR   CHKCOM
320:    033F 20 8B D0            JSR   GETVAR   ; VARIABLE HOLEN
325:    0342 48                  PHA
325:    0343 98                  TYA
325:    0344 48                  PHA
330:    0345 20 A3 D6            JSR   FRESTR
340:    0348 20 FD CE            JSR   CHKCOM
350:    034B 20 9E D7            JSR   GETBYT   ; LAENGE IN X-REGISTER
365:    034E 68                  PLA
365:    034F 85 65               STA   DESCRPT + 1
367:    0351 68                  PLA
367:    0352 85 64               STA   DESCRPT
368:    0354 8A                  TXA            ; LAENGE
370:    0355 48                  PHA            ; LAENGE AUF STACK
380:    0356 20 75 D4            JSR   SETSTR   ; PLATZ FUER STRING RESERVIEREN
381:    0359 84 FB               STY   TEMP
381:    035B A0 00               LDY   #0
381:    035D 91 64               STA   (DESCRPT),Y
381:    035F C8                  INY
381:    0360 8A                  TXA
```

```
381:    0361 91 64              STA  (DESCRPT),Y
381:    0363 C8                 INY
382:    0364 A5 FB              LDA  TEMP
382:    0366 91 64              STA  (DESCRPT),Y
390:    0368 20 FD CE           JSR  CHKCOM
395:    036B A0 01              LDY  #1
395:    036D B1 64              LDA  (DESCRPT),Y
395:    036F 48                 PHA
395:    0370 C8                 INY
395:    0371 B1 64              LDA  (DESCRPT),Y
395:    0373 48                 PHA
400:    0374 20 9E D7           JSR  GETBYT   ; FUELLZEICHEN HOLEN
410:    0377 68                 PLA
410:    0378 85 65              STA  DESCRPT+1
410:    037A 68                 PLA
410:    037B 85 64              STA  DESCRPT
420:    037D 68                 PLA
420:    037E A8                 TAY           ; LAENGE IN Y-REGISTER
430:    037F 8A                 TXA           ; FUELLZEICHEN IN AKKU
440:    0380 88         LOOP    DEY
440:    0381 91 64              STA  (DESCRPT),Y  ; STRING ERSTELLEN
450:    0383 D0 FB              BNE  LOOP
460:    0385 60                 RTS
```

6.6 Erweiterung des MID$-Befehls

Das Programm, das wir Ihnen nun vorstellen wollen, stellt eine sehr nützliche Erweiterung des MID$-Befehls dar.
Wenn Sie bisher ein oder mehrere Zeichen in einem String ändern wollten, hatten Sie keinen anderen Weg, als eine umständliche Konstruktion über LEFT$ und RIGHT$ zu nehmen. Ein Beispiel dazu:

Nehmen wir an, Sie haben folgenden String :

```
A$="ABCDEFGGJKLMN"
```

Sie möchten nun den achten Buchstaben in ein 'I' ändern. Dies müßten Sie so machen:

```
A$ = LEFT$(A$,7)+"I"+RIGHT$(A$,6)
```

Dies ist ja noch eine recht einfache Formel. Kompliziert wird die Angelegenheit, wenn Sie diese Formel allgemeingülig auch für mehrere Zeichen machen wollen. Nehmen wir an, der zu verändernde String steht in A$, der einzusetzende String in B$, die Position in PO. Die Formel sieht dann folgendermaßen aus:

```
A$ = LEFT$(A$,PO-1)+B$+RIGHT$(A$,LEN(A$)-LEN(B$)-PO+1)
```

Diese Formel sieht schon etwas mächtiger aus. Stellen Sie sich nun noch vor, Sie haben indizierte Variablen und Sie werden Schwierigkeiten bekommen, dies alles in eine Zeile zu packen. Und dies alles nur, weil ein Ausdruck wie

```
MID$(A$,PO-1,LEN(B$))=B$
```

nicht erlaubt ist! Diesem übel werden wir nun schnellstens Abhilfe schaffen. Auf den nächsten beiden Seiten finden Sie das Assemblerlisting zu dem Programm. Dieses Programm muß lediglich durch SYS 828 einmal initialisiert werden. Dann sind Befehle wie der Obenstehende erlaubt.

ε

```
90:     033C                      .OPT P1
                        ; MID$ ALS PSEUDOVARIABLE
                        ;
                        ; MID$(STRINGVARIABLE,POSITION,LAENGE) = STRINGAUSDRUCK
                        ; MID$(STRINGVARIABLE,POSITION) = STRINGAUSDRUCK
                        ;
200:    033C            MIDCODE   =    $CA
210:    033C            EXECUT    =    $308     ; VECTOR FUER STATEMENT AUSFUEHREN
220:    033C            CHRGET    =    $73
230:    033C            CHRGOT    =    CHRGET + 6
240:    033C            EXECOLD   =    $C7E7
250:    033C            VARNAM    =    $45
255:    033C            VARADR    =    $49
260:    033C            DESCRPT   =    $64
270:    033C            TESTSTR   =    $CD8F
280:    033C            GETVAR    =    $D08B
290:    033C            SETSTR    =    $CA52
300:    033C            CHKAUF    =    $CEFA    ; KLAMMER AUF
310:    033C            CHKZU     =    $CEF7    ; KLAMMER ZU
320:    033C            CHKCOM    =    $CEFD    ; KOMMA
325:    033C            TEST      =    $CEFF
330:    033C            GETBYT    =    $D79E
340:    033C            FRMEVL    =    $CD9E
350:    033C            ILLQUAN   =    $D248
352:    033C            FRESTR    =    $D6A3
355:    0003                      *=   3
360:    0004            LAENGE    *=   *+1
370:    0005            POSITION  *=   *+1
372:    0007            VARSTR    *=   *+2
375:    0007            GLEICH    =    $B2
378:    0007            ZEIG2     =    $50
                        ;
390:    033C                      *=   828
400:    033C A9 47      INIT      LDA  #<MIDTEST
410:    033E A0 03                LDY  #>MIDTEST
420:    0340 8D 08 03             STA  EXECUT
430:    0343 8C 09 03             STY  EXECUT+1
440:    0346 60                   RTS
450:    0347 20 73 00   MIDTEST   JSR  CHRGET
460:    034A C9 CA                CMP  #MIDCODE   ; CODE FUER MID$
470:    034C F0 06                BEQ  MIDSTR   ; JA
480:    034E 20 79 00             JSR  CHRGOT
```

```
490:    0351 4C E7 C7          JMP  EXECOLD  ; NORMALES STATEMENT AUSFUEHREN
500:    0354 20 73 00 MIDSTR   JSR  CHRGET  ; NAECHSTES ZEICHEN
505:    0357 20 FA CE          JSR  CHKAUF  ; KLAMMER AUF
510:    035A 20 8B D0          JSR  GETVAR  ; VARIABLE HOLEN
520:    035D 85 64             STA  DESCRPT
530:    035F 84 65             STY  DESCRPT+1
535:    0361 85 49             STA  VARADR
535:    0363 84 4A             STY  VARADR+1
540:    0365 20 A3 D6          JSR  FRESTR
545:    0368 A0 00             LDY  #0
545:    036A B1 64             LDA  (DESCRPT),Y
545:    036C 48                PHA          ; LAENGE
545:    036D F0 2E             BEQ  ILL
550:    036F 20 52 CA          JSR  SETSTR  ; STRING IN RAM UEBERTRAGEN
560:    0372 A0 01             LDY  #1
560:    0374 B1 49             LDA  (VARADR),Y
560:    0376 85 05             STA  VARSTR  ; VARIABLENADRESSE MERKEN
570:    0378 C8                INY
570:    0379 B1 49             LDA  (VARADR),Y
570:    037B 85 06             STA  VARSTR+1
600:    037D 20 FD CE          JSR  CHKCOM
610:    0380 20 9E D7          JSR  GETBYT  ; POSITION HOLEN
620:    0383 8A                TXA
630:    0384 F0 17             BEQ  ILL
650:    0386 CA                DEX
650:    0387 86 04             STX  POSITION
660:    0389 20 79 00          JSR  CHRGOT
660:    038C C9 29             CMP  #")"    ; AUSDRUCK ZU ENDE
665:    038E D0 04             BNE  NEXT
665:    0390 A9 FF             LDA  #$FF    ; MAX. LAENGE
665:    0392 D0 0C             BNE  STORE
670:    0394 20 FD CE NEXT     JSR  CHKCOM
670:    0397 20 9E D7          JSR  GETBYT  ; LAENGE HOLEN
680:    039A 8A                TXA
690:    039B D0 03             BNE  *+5
700:    039D 4C 48 D2 ILL      JMP  ILLQUAN
710:    03A0 85 03    STORE    STA  LAENGE
```

```
715:    03A2 68                 PLA
715:    03A3 38                 SEC
715:    03A4 E5 04              SBC  POSITION
717:    03A6 C5 03              CMP  LAENGE
717:    03A8 B0 02              BCS  OK
717:    03AA 85 03              STA  LAENGE
720:    03AC 20 F7 CE OK        JSR  CHKZU     ; KLAMMER ZU
730:    03AF A9 B2              LDA  #GLEICH
770:    03B1 20 FF CE           JSR  TEST
780:    03B4 20 9E CD           JSR  FRMEVL    ; AUSDRUCK HOLEN
790:    03B7 20 A3 D6           JSR  FRESTR
800:    03BA A0 02              LDY  #2
800:    03BC B1 64              LDA  (DESCRPT),Y
800:    03BE 85 51              STA  ZEIG2+1
800:    03C0 88                 DEY
800:    03C1 B1 64              LDA  (DESCRPT),Y
800:    03C3 85 50              STA  ZEIG2
810:    03C5 88                 DEY
810:    03C6 B1 64              LDA  (DESCRPT),Y
820:    03C8 F0 D3              BEQ  ILL       ; NULL DANN FEHLER
840:    03CA C5 03              CMP  LAENGE
850:    03CC B0 02              BCS  OK1
860:    03CE 85 03              STA  LAENGE
870:    03D0 A5 05     OK1      LDA  VARSTR
880:    03D2 18                 CLC
880:    03D3 65 04              ADC  POSITION
910:    03D5 85 05              STA  VARSTR
910:    03D7 90 02              BCC  *+4
920:    03D9 E6 06              INC  VARSTR+1
940:    03DB A4 03              LDY  LAENGE
950:    03DD 88        LOOP     DEY
950:    03DE B1 50              LDA  (ZEIG2),Y  ; ZEICHEN AUS STRINGAUSDRUCK
960:    03E0 91 05              STA  (VARSTR),Y  ; IN STRINGVARIABLE UEBERTRAGEN
970:    03E2 C0 00              CPY  #0
970:    03E4 D0 F7              BNE  LOOP
980:    03E6 4C AE C7           JMP  $C7AE     ; ZUR INTERPRETERSCHLEIFE
```

6.7 Der POP-Befehl

In vielen BASIC-Dialekten gibt es den sogenannten POP-Befehl. Er ermöglicht es, ein Unterprogramm zu verlassen, ohne den 'Return'-Befehl zu benutzen. Dies ist zwar auch ohne den POP-Befehl grundsätzlich möglich, hat jedoch den unschönen Nebeneffekt, daß die Rücksprungadressen der GOSUB-Routine nicht vom Stack entfernt werden. Wenn Sie oft auf diese Weise aus einem Unterprogramm heraus springen, wird die Folge ein 'OUT OF MEMORY ERROR' sein. Der Grund dafür ist, daß der Stack übergelaufen ist.
Das folgende Programm tut nichts anderes, als 5 Bytes vom Stack zu entfernen. Rufen Sie dieses Programm jedesmal auf, bevor Sie ein Unterprogramm mit einem 'GOTO'-Befehl verlassen.

```
                       ;
                       ; POP-BEFEHL
                       ;
                       ; AUFRUF MIT SYS 828
                       ;
                       ;
033C                          .OPT P1
033C                          *=   828
033C 68                       PLA
033D 68                       PLA
033E A9 FF                    LDA  #255
0340 85 4A                    STA  $4A
0342 20 8A C3                 JSR  $C38A
0345 C9 8D                    CMP  #141
0347 F0 05                    BEQ  GEFUNDEN
0349 A2 0C                    LDX  #12
034B 4C 37 C4                 JMP  $C437
034E 9A              GEFUNDEN TXS
034F 68                       PLA
0350 68                       PLA
0351 68                       PLA
0352 68                       PLA
0353 68                       PLA
0354 4C AE C7                 JMP  $C7AE
```

KAPITEL 7 : ANWENDUNGSPROGRAMMIERUNG FÜR FORTGESCHRITTENE

In diesem Kapitel wollen wir Ihnen eine Reihe von Tips der Programmierung - an Hand von ausführlichen Beispielsprogrammen - geben. Diese Programme können von Ihnen eingegeben werden und verschiedene Aufgaben lösen. Das erste Programm ist zum Beispiel ein Programm zur Lagerhaltung. Jeder, der beispielsweise ein kleines Geschäft hat, kann mit diesem Programm seinen Steuerberater etwas entlasten. Es berechnet die Anzahl der einzelnen Artikel, ihre Einkaufs- und Verkaufspreise und natürlich die Mehrwertsteuer. So wissen Sie auf Knopfdruck "was die Stunde geschlagen hat". Danach beschreiben wir das Prinzip der Textverarbeitung. Hier können wir nur die Grundzüge dieses wohl aufwendigsten Thema der EDV erläutern. Aber Sie werden schnell hinter die Idee des Texteinlesens, Analysieren und Ausgeben kommen. Am Schluß dieses Kapitels finden Sie ein weiteres Programm. Diesmal was für die Spielfreudigen unter den Lesern. Suchen Sie Gold in einem Bergwerk - passen Sie aber auf, daß Sie nicht in einem Wassereinbruch geraten, oder durch einen Stollen fallen.

All diese Programme sollen zwar dem Anwender erst einmal Freude an einem fertigen Programm bringen - sie sind in erster Linie dazu gedacht, den interessierten Anwender die Möglichkeiten von Dateiverwaltung, Tastaturabfrage, Farbe, Graphik usw. des VC-20 verdeutlichen. Daher haben wir uns bemüht, jedes Programm so ausführlich wie möglich zu beschreiben. Sollten Sie denoch Fragen haben, so wenden Sie sich doch an uns. Aber zunächst sollten Sie erst einmal versuchen zu verstehen, was wir da programmiert haben. Ein bisschen Ahnung von BASIC gehört zwar schon dazu - aber auch der absolute Anfänger kann an Hand dieser Programme sehr viel nützliches lernen.

7.1 Der Umgang mit Daten

Ein sehr wichtiges Thema in der EDV ist auf jedem Gerät, ob groß, ob klein, der Umgang mit Daten und Dateien. Wir haben nun ein Programm entworfen, das Ihnen einen Großteil an Tips und Tricks bei der Programmierung einer Dateiverwaltung demonstriert. Dieses Programm erlaubt es verschiedene Artikel zu erfassen, die Einkaufs- oder Verkaufszahlen einzugeben und danach die Summen auszugeben. Dieses Programm benötigt, um ordnungsgemäß zu laufen, die 16K Erweiterung. Außerdem wird eine Floppy-Station benötigt und der Seikosha Drucker GP/100 VC.

Nun aber zuerst etwas über die Möglichkeiten der verschiedenen Dateiverwaltungsarten. Die bekannteste Art und Weise Daten zu lesen und zu schreiben, ist die sequentielle Datei. Dieses Verfahren, ist übernommen wurden, aus der Zeit, als auch die größeren Computer hauptsächlich mit Bandlaufwerken benutzt wurden. Um den Unterschied zwischen Platten- und Bandlaufwerken zu begreifen, stellen Sie sich am besten einen normalen Kassettenrecorder und auf der anderen Seite den Plattenspieler vor. Um nun an ein bestimmtes Musikstück zu gelangen, so ist es bei einem Bandlaufwerk notwendig, erst einmal an den Anfang des Bandes zu Spulen. Dann wird das Zählwerk auf 0 gestellt und es kann begonnen werden, an die Stelle zu fahren, an der das Musikstück beginnt (wir gehen davon aus, daß die Zählwerksziffern bei jedem neuen Musikstück bekannt sind). Erst dann kann man der Musik lauschen. Bei einem Platenspieler dagegen, haben wir die Möglichkeit den Plattenarm direkt auf den Anfang des Musikstückes zu setzen - sofern man die Rillen sehen kann.
Dieses Verständnis kann man auch bei Datasette und Floppy anwenden. Es ist klar, daß man bei einer Datasette nur Date für Date lesen kann - also sequentiell - nicht aber einen speziellen Satz. über das Lesen und Schreiben von Daten mit Hilfe des Kassettenrecorders, können Sie in dem entsprechenden Kapitel in diesem Buch nachlesen.

Wir wollen uns an dieser Stelle nur mit der Floppy beschäftigen. Denn wer richtige Datenverwaltung betreiben will, der kommt um dieses schnelle Speichermedium nicht herum.

Auch bei der Floppy müssen wir, wie schon oben erwähnt, zwischen sequentiellen und relative Datei unterscheiden. Den Begriff "sequentiell" haben wir schon beschrieben - es beduetet eben: nacheinander. Was aber bedeutet nun "relativ". Dieses Wort kennen wir schon aus unserem normalen Wortschatz. Direkt umgesetzt auf die EDV bedeutet dies: abhängig von der momentanen Position.

Wir haben eben gesehen, daß wir beim Bandlaufwerk erst an den Anfang spulen mußten, um einen bestimmten "Satz", also eine Menge von Daten, zu lesen. Eine relative Datei ist hier viel flexibler. Bei ihr ist es möglich, abhängig von der momentanen Position des Lesekopfes, Sätze zu lesen. Verdeutlicht an einem Beispiel:

Auf einer Diskette befinden sich 100 Sätze. Aus dieser Menge wollen wir nun den 78ten Satz lesen. Bei einer sequentiellen Datei hieße das, erst alle 77 davor liegenden Sätze zu lesen, um dann endlich an den 78ten Satz zu gelangen. Was das für ein Zeitaufwand bedeutet, können Sie sich wohl vorstellen. Denken wir dann noch daran, mehrere Sätze zu vergleichen oder zu sortieren, so wäre dies für einen schnellen Programmablauf völlig ungeeignet. Zu diesem Schluß kamen auch die verschiedenen Computerhersteller. Das Ergebnis ist die relative Datei. Dort kann ich nun einfach sagen: Lese den 78ten Satz. Der Computer berechnet nun selbstständig, um wieviele Sätze vor- oder zurückgeblättert werden muß, um diesen Satz zu lesen. Durch diese revolutionäre Erfindung, war es nun möglich den Zeitfaktor, um einen bestimmten Satz zu lesen, auf ein Minimum zu reduzieren.

Beim VC-20 gibt es allerdings ein paar Sachen, auf die bei der Programmierung von Dateien zu achten ist. So verfügt das BASIC des VC-20, im Unterschied zu dem BASIC-4 der großen Commodore Computer, nicht über Befehle, die es ermöglichen relative Dateien zu verwalten.

Glücklicherweise gibt es ein paar Tricks, die es ermöglichen, trotzdem mit relativen Dateien zu arbeiten. So verfügt ja

bekanntlich die Floppy über ein eigenes Betriebssystem. Das Problem besteht eigentlich nur darin, Befehle, die die relativen Dateien verwalten, dem Floppy mitzuteilen. Dazu dient aber der Kommando (oder Befehls-) Kanal des VC-20 (Kanal 15).

Zunächst wird die Datei, von der Daten gelesen oder in die Daten geschrieben werden sollen, geöffnet. Von BASIC sieht das so aus:

```
OPEN Datei#, Geräte#, Kanal#,"NAME,L,"+CHR$(Länge)
```

oder an einem konkreten Beispiel (s. Programm):

```
OPEN 1,8,1,"0:ARTDAT,L,"+CHR$(64)
```

Dieser Befehl öffnet die Datei 1, mit dem Namen "ARTDAT", der Gerätenummer 8 (Floppy) und dem Kanal 1. Das Wichtigste an diesem Befehl ist aber die Länge der Datei. Sie ist in unserem Beispiel mit 64 angegeben. Anders als bei den sequentiellen Dateien, so ist es bei relativen datei, äußerst wichtig die Länge der Datei mit anzugeben. Durch Angabe dieser Länge, ist es dem Computer erst möglich, die relativen "Sprünge" zu den Sätzen zu berechnen. Bei der sequentiellen Datei kann diese Angabe entfallen, da hier die Länge unerheblich ist.

Nun weiß die Floppy 1540/1541, daß eine relative Datei mit der Länge 64 und dem Namen "ARTDAT" angelegt werden soll. Wenn wir nun einen bestimmten Datensatz schreiben wollen, so müssen wir nun noch die Stelle angeben an der der Satz gespeichert werden soll. Von BASIC aus könnte die Position des Zeigers so aussehen:

```
PRINT# Kanal#,"P";CHR$(Kanal#);CHR$(Satz#-low);CHR$(Satz#-high)
```

Stellen wir uns nun vor, daß wir in den 78ten Satz schreiben wollen. So müssen wir nun zunächst Satz#-low und Satz#-high berechnen. Dies geschieht mit folgender Formel:

```
HB=INT(RN/256): LB=RN-HB*256
```

oder an dem konkreten Beispiel:

HB=INT(78/256): LB=78-HB*256

und als Ergebnis für Satz#-low (LB) und Satz#-high (HB):

HB=0: LB=78

Eingesetzt in unsere obige Formel zum Positionieren der Zeiger lautet demanch so:

PRINT#1,"P";CHR$(1);CHR$(0);CHR$(78)

nun steht der Zeiger an der richtigen Stelle und wir können den Satz schreiben. Dieses geschieht mit dem PRINT#-Befehl. Alle Variablen, die in die Datei gespeichert werden sollen, müssen mit einem CHR$(13), also einem RETURN, voneinander getrennt werden. Den letzten RETURN setzt dann der Computer selber. Siehe hierzu die Zeilen 10070-10080 des Beipsielprogrammes.

Das Lesen eines Satzes geschieht nach dem selben Muster. Zunächst wird der Zeiger positioniert, dann die Routine zum Lesen aufgerufen. Hier werden dann jweils die Variablen, die beim Schreiben durch RETURNs getrennt wurden, einzelnen eingelesen. Der VC-20 hält hierfür den INPUT#-Befehl bereit. In unserem Beispiel erfolgt das Lesen in den Zeilen 10030 bis 10060.

Eine weitere wichtige Einrichtung für die Benutzung der Dateien, und generell der Floppy, ist der Fehlerkanal. Wenn Sie einen Datei geöffnet haben, sollten Sie immer zuerst den Fehlerkanal abfragen, um so rechtzeitig festzustellen, ob alles in Ordung ist (s. Zeilen 12000-12050).

Mit Hilfe dieser Routinen ist es nun möglich, beliebige, relative Dateien zu erzeugen und zu verarbeiten. Sie können nun zum Beispiel Ihre eigene Adressenverwaltung, oder Ihre

Schallplattensammlung, verwalten. Nehmen Sie dazu einfach unser Programm als Anregung. Wenn Sie dieses Programm in seinen Grundzügen verstanden haben, dann wird Ihnen auch eine eigene Entwicklung keine Schwierigkeiten mehr bereiten. Die Dateiverwaltung zählt twar nicht unbedingt zu den einfachsten, aber auf jeden Fall zu den interessantesten Aufagen in der Programmierung.
Nun aber noch etwas zu unserem Programm. Nachdem Sie es durch RUN gestartet haben, erscheint die Frage, ob das Diskettengerät bereit ist. Diese Meldung wird solange erscheinen, bis Sie die 'J'-Taste drücken. Danach werden Sie gefragt, ob Sie eine neue Diskette verwenden woll. Neu heißt im dem Fall nicht nur unformatiert, sondern auch, daß noch keine Datei mit dem Namen ARTDAT vorhanden ist. Aber Vorsicht, die Diskette wird formatiert. Benutzen Sie also nicht eine Ihrer Programmdisketten. Wenn dieser Vorgang beendet ist, erscheint das Hauptmenü. Hier können Sie nun eine der 6 Möglichen Funktionen aufrufen.

Wenn Sie einen Datensatz gestalten wollen, so denken Sie daran, daß die Eingabe nicht länger sein darf, als in den DATA-Zeilen (30-82) festgelegt ist. Diese Zeilen sind do aufgebaut, daß zuerst der Name und dann die Länge der Bezeichnung aufgeführt wird. Wollen Sie einen bereits eingegebenen Datensatz wieder löschen, so gehen Sie in die Routine AENDERN, und schreiben als erstes Zeichen der Bezeichnung Warenart ein Klammeraffen (@). Somit wird dieser Satz als gelöscht gekennzeichnet.

Wenn Sie in irgendeinem Unterprogramm aufgefordert werden, eine Artikelnummer einzugeben, so können Sie durch die Eingabe von "ENDE" wieder ins Haupt- oder übermenü zurückgelangen.

Die Zeilen 11 bis 14 stellen den heutigen Mehrwertsteuersatz dar. ME(1) und ME(2) sind die beiden alten Sätze von 6.5% und 13%. Die nächsten beiden sind die neuen Sätze von 7% und 14%. Werden Sie also nach dem Mehrwertsteuersatz gefragt, so geben Sie einfach eine 1 für 6.5%, eine 2 für 13%, eine 3 für 7% und eine 4 für 14% ein. Der Computer berechnet sich dann den korrekten

Mehrwetsteuersatz.

Eine Besonderheit ist bei diesem Programm aber noch zu beachten. Artikelnummern, die von 800 bis 999 gehen (999 ist die größte mögliche Nummer), werden nicht als Artikel gewertet sondern als Ausgaben oder Einnahmen. So können Sie gleichzeitig zu den verschiedenen Artikeln auch noch Ihre Privatkonten, Bankkonten usw. überwachen. Die Eingabe dieser Konten erfolgt im Unterprogramm EINGABE DER KASSENZETTEL. Wenn die erste Eingabe, also der erste "Kassenzettel", genauer gesagt die Rechnung, eine Zahl größer 799 war, so erwartet der Computer weitere Eingaben dieser Art. Das heißt, daß Sie nicht abwechselnd Artikel und Rechnungen eingeben können, sondern diese nur jeweils getrennt.

Außerdem ist darauf zu achten, daß bei der Arbeit mit diesem Programm, Drucker und Floppy immer angeschaltet sind. Haben Sie aber keinen Drucker, so müssen Sie das Programm umschreiben. Sie sehen ja an den PRINT#4-Befehlen, in welchen Zeilen eine Ausgabe auf dem Drucker erfolgen soll.

Wenn Sie noch weitere Filialen oder Personen mit diesem Programm unterstützen wollen, so empfiehlt es sich für jede Filiale oder Person, eine neue Diskette anzulegen. Dann wissen Sie zu jeder Zeit über die finanzielle Lage bescheid.

Wir hoffen, daß Sie mit Hilfe deises Programmes einen Einblick in die Datenverarbeitung, und besonders in die Datenverarbeitung mit relativen Dateien bekommen haben. Es sieht auf den ersten Blick schwieriger aus als es ist. Mit ein wenig übung, werden auch Sie rasch solche Programme selbst entwerfen können.

Noch eine Anmerkung zum Dateiverwaltungsprogramm. Bei einer Neuerstellung einer Diskette, beginnt nach dem letzen Zugriff auf die Diskette die rote LED der Floppy zu blinken. Dies ist jedoch kein Fehler, sondern nur eine besondere Reaktion der Floppy 1540/1541 bei der Benutzung einer relativen Datei.

Wir wollen ja eine Datei mit maximal 999 Einträgen erstellen. Dazu haben wir mit dem Befehl

```
230 OPEN 15,8,15,"N:DATENDISKETTE,VC"
240 OPEN 1,8,3,"0:ARTDAT,L,"+CHR$(64)
250 PRINT#1,CHR$(255);
```

eine entsprechende Datei erstellt. Allerdings wird beim Schreiben des CHR$(255) auf Diskette als letzter Satz ein

RECORD NOT PRESENT

Fehler erkannt. Dies hat jedoch keinen Einfluss auf Ihre Datei.

Im übrigen können im Zusammenhang mit der Dateiverwaltung noch andere Fehlermeldungen entstehen. Dies sind unter anderem:

RECORD NOT PRESENT

OVERFLOW IN RECORD

FILE TOO LARGE

Die Bedeutung der jeweiligen Fehler können Sie im Handbuch der Floppy nachlesen.

Hier nun das oben besprochene Programm zur Dateiverwaltung auf dem VC-20.

```
10 CLR
11 ME(1)=1.065
12 ME(2)=1.13
13 ME(3)=1.07
14 ME(4)=1.14
20 FOR I=1 TO 7
25 READ TD$(I),TD(I)
27 NEXT I
30 DATA "1) ARTIKEL NR.",3
32 DATA "2) WARENART",20
34 DATA "3) ANZAHL",3
36 DATA "4) EINZELPREIS EK",7
38 DATA "5) GESAMTPREIS EK",8
40 DATA "6) EINZELPREIS VK",7
42 DATA "7) GESAMTPREIS VK",8
50 FOR I=1 TO 3
52 READ TI$(I),TI(I)
54 NEXT I
56 DATA "1) ARTIKEL NR.",3
58 DATA "2) WARENART",20
60 DATA "3) MWST GRUPPE",1
70 FOR I=1 TO 4
72 READ TT$(I),TT(I)
74 NEXT I
76 DATA "1) FILIALE NR.",1
78 DATA "2) DATUM",8
80 DATA "3) RECHNUNGS NR.",8
82 DATA "4) BELEG NR.",8
100 PRINT CHR$(147);
110 PRINT "**********************";
120 PRINT "*ARTIKELVERWALTUNG VC*";
130 PRINT "**********************";
140 PRINT: PRINT
```

```
150 PRINT "DISK BEREIT (J/N)? ";
160 GET A$: IF A$="" THEN 160
170 IF A$<>"J" THEN 160
180 PRINT A$
185 PRINT
190 OPEN 15,8,15,"IO"
195 CLOSE 15
200 PRINT "NEUE DISKETTE (J/N)? ";
210 GET A$: IF A$="" THEN 210
220 IF A$<>"J" THEN 300
222 PRINT A$
224 PRINT
230 OPEN 15,8,15,"N:DATENDISKETTE,VC"
240 OPEN 1,8,3,"0:ARTDAT,L,"+CHR$(64)
250 PRINT#15,"P"+CHR$(3)+CHR$(231)+CHR$(3)+CHR$(1)
260 PRINT#1,CHR$(255);
270 RM=INT(167132/64)
280 CLOSE 1
290 CLOSE 15
300 PRINT CHR$(147);
310 PRINT "**********************";
320 PRINT "*ARTIKELVERWALTUNG VC*";
330 PRINT "**********************";
340 PRINT: PRINT
345 PRINT "      HAUPTMENU": PRINT
350 PRINT "1) EINGABE DER DATEI": PRINT
355 PRINT "2) AENDERN DER DATEI": PRINT
360 PRINT "3) EINGABE KASSENZ.": PRINT
365 PRINT "4) DRUCKEN ART.-LISTE": PRINT
370 PRINT "5) DRUCKEN AUSWERTUNG": PRINT
375 PRINT "6) PROGRAMMENDE": PRINT: PRINT
380 PRINT "BITTE WAEHLEN SIE ";
390 GET A$: IF A$="" THEN 390
400 A=VAL(A$): IF A<1 OR A>6 THEN 390
410 PRINT A$;
420 FOR I=1 TO 1000: NEXT
430 ON VAL(A$) GOTO 1000,2000,3000,4000,5000,6000
```

```
1000 OPEN 15,8,15: OPEN 8,8,8,"0:ARTDAT"
1002 GOSUB 12000
1005 PRINT CHR$(147);
1010 PRINT "***********************";
1020 PRINT "*ARTIKELVERWALTUNG VC*";
1030 PRINT "***********************";
1040 PRINT: PRINT
1050 PRINT "  EINGABE DER DATEI": PRINT: PRINT
1060 FOR I=1 TO 3
1065 TE$(I)=""
1070 PRINT TI$(I)
1080 INPUT TE$(I)
1090 PRINT
1092 IF TE$(1)="ENDE" THEN 1200
1095 IF LEN(TE$(I))>TI(I) THEN 1065
1100 NEXT I
1102 FOR I=4 TO 8
1104 TE$(I)=""
1106 NEXT I
1110 RN=VAL(TE$(1))
1120 IF RN<1 OR RN>999 THEN 1005
1130 GOSUB 10000
1140 GOSUB 10070
1150 GOTO 1005
1200 CLOSE 8
1210 CLOSE 15
1220 GOTO 300
2000 OPEN 15,8,15: OPEN 8,8,8,"0:ARTDAT"
2002 GOSUB 12000
2005 PRINT CHR$(147);
2010 PRINT "***********************";
2020 PRINT "*ARTIKELVERWALTUNG VC*";
2030 PRINT "***********************";
2040 PRINT: PRINT
2050 PRINT "  AENDERN DER DATEI": PRINT: PRINT
2055 TE$(1)=""
2060 PRINT TI$(1)
```

```
2070 INPUT TE$(1)
2080 PRINT
2090 IF TE$(1)="ENDE" THEN 2400
2100 IF LEN(TE$(1))>TI(1) THEN 2055
2110 RN=VAL(TE$(1))
2120 IF RN<1 OR RN>999 THEN 2005
2130 GOSUB 10000
2140 GOSUB 10030
2142 IF VAL(TE$(1))<>RN THEN 2005
2150 PRINT CHR$(147);
2160 PRINT "**********************";
2170 PRINT "*ARTIKELVERWALTUNG VC*";
2180 PRINT "**********************";
2190 PRINT: PRINT
2200 PRINT "  AENDERN DER DATEI": PRINT: PRINT
2210 FOR I=1 TO 3
2220 PRINT TI$(I)
2230 PRINT "? ";TE$(I)
2240 PRINT CHR$(145);
2260 INPUT TE$(I)
2270 PRINT
2280 IF TE$(1)="ENDE" THEN 2400
2290 IF LEN(TE$(I))>TI(I) THEN 2250
2300 NEXT I
2310 RN=VAL(TE$(1))
2320 IF RN<1 OR RN>999 THEN 2005
2330 GOSUB 10000
2340 GOSUB 10070
2350 GOTO 2005
2400 CLOSE 4
2410 CLOSE 8
2420 CLOSE 15
2430 GOTO 300
2530 GOTO 3005
3000 OPEN 15,8,15: OPEN 8,8,8,"0:ARTDAT"
3001 GOSUB 12000
3002 DV=1
```

```
3003 GOSUB 8000
3004 GOSUB 8500
3005 PRINT CHR$(147);
3010 PRINT "**********************";
3020 PRINT "*ARTIKELVERWALTUNG VC*";
3030 PRINT "**********************";
3040 PRINT: PRINT
3050 PRINT " EINGABE KASSENZETTEL": PRINT: PRINT
3060 TE$(1)=""
3070 PRINT TI$(1)
3080 INPUT TE$(1)
3090 PRINT
3100 IF TE$(1)="ENDE" THEN 3700
3110 IF LEN(TE$(1))>TI(1) THEN 3060
3120 RN=VAL(TE$(1))
3130 IF RN<1 OR RN>999 THEN 3005
3132 IF DW=1 AND RN>799 THEN 3005
3134 IF DW=2 AND RN<800 THEN 3000
3140 GOSUB 10000
3150 GOSUB 10030
3152 IF VAL(TE$(1))<>RN THEN 3005
3160 PRINT CHR$(147);
3170 PRINT "**********************";
3180 PRINT "*ARTIKELVERWALTUNG VC*";
3190 PRINT "**********************";
3200 PRINT: PRINT
3210 PRINT " EINGABE KASSENZETTEL": PRINT: PRINT
3212 FOR I=1 TO 5
3214 TH$(I)=TE$(I+3)
3216 NEXT I
3220 FOR I=1 TO 2
3230 PRINT TD$(I)
3235 PRINT "? ";TE$(I)
3237 TX$(I)=TE$(I)
3240 PRINT
3250 NEXT I
3255 TX$(3)=TE$(3)
```

```
3260 PRINT TD$(3)
3270 INPUT TX$(4)
3275 TE$(4)=TX$(4)
3280 PRINT
3285 IF VAL(TE$(4))<-999 OR VAL(TE$(4))>999 THEN 3260
3287 IF LEN(TE$(4))>TD(3) THEN 3260
3290 PRINT TD$(4)
3295 TE$(5)=""
3300 INPUT TX$(5)
3305 TE$(5)=TX$(5)
3310 PRINT
3315 IF LEN(TE$(5))>TD(4) THEN 3290
3320 PRINT TD$(6)
3325 TE$(7)=""
3330 INPUT TX$(7)
3335 TE$(7)=TX$(7)
3340 PRINT
3345 IF LEN(TE$(7))>TD(6) THEN 3320
3350 TH=VAL(TE$(5))*VAL(TE$(4))
3351 TX$(6)=STR$(TH)
3352 TH=TH+VAL(TH$(3))
3355 TE$(6)=STR$(TH)
3360 TH=VAL(TE$(7))*VAL(TE$(4))
3361 TX$(8)=STR$(TH)
3362 TH=TH+VAL(TH$(5))
3365 TE$(8)=STR$(TH)
3370 TH=VAL(TE$(5))
3371 IF VAL(TE$(4))<1 THEN TH=-TH
3372 TH=TH+VAL(TH$(2))
3375 TE$(5)=STR$(TH)
3380 TH=VAL(TE$(7))
3381 IF VAL(TE$(4))<1 THEN TH=-TH
3382 TH=TH+VAL(TH$(4))
3385 TE$(7)=STR$(TH)
3390 TH=VAL(TE$(4))
3392 TH=TH+VAL(TH$(1))
3395 TE$(4)=STR$(TH)
```

```
3460 RN=VAL(TE$(1))
3470 GOSUB 10000
3480 GOSUB 10070
3485 FOR I=1 TO 8
3486 TE$(I)=TX$(I)
3487 TX$(I)=""
3488 NEXT I
3490 IF DW=0 AND VAL(TE$(1))<800 THEN DW=1: GOSUB 5360: GOTO 3510
3500 IF DW=0 AND VAL(TE$(1))>799 THEN DW=2: GOSUB 7005: GOTO 3520
3510 IF DW=1 THEN GOSUB 5520: GOTO 3530
3520 IF DW=2 THEN GOSUB 7120
3530 GOTO 3005
3700 IF DW=1 THEN GOSUB 5590: GOTO 3800
3710 IF DW=2 THEN GOSUB 7190
3800 DW=0: DV=0
3999 CLOSE 4: CLOSE 8: CLOSE 15: GOTO 300
4000 OPEN 15,8,15: OPEN 8,8,8,"0:ARTDAT"
4002 GOSUB 12000
4005 PRINT CHR$(147);
4010 PRINT "**********************";
4020 PRINT "*ARTIKELVERWALTUNG VC*";
4030 PRINT "**********************";
4040 PRINT: PRINT
4050 PRINT " DRUCKEN ARTIKELLISTE": PRINT: PRINT
4060 FOR I=1 TO 2
4070 TE$(I)=""
4080 PRINT TT$(I)
4090 INPUT TE$(I)
4100 PRINT
4110 IF TE$(1)="ENDE" THEN 4999
4120 IF LEN(TE$(I))>TT(I) THEN 4070
4130 NEXT I
4135 TE$(3)="": TE$(4)=""
4140 IF DV=1 THEN RETURN
4200 PRINT CHR$(147);
4210 PRINT "**********************";
4220 PRINT "*ARTIKELVERWALTUNG VC*";
```

```
4230 PRINT "**********************";
4240 PRINT: PRINT
4250 PRINT " DRUCKEN ARTIKELLISTE": PRINT: PRINT
4260 PRINT "DRUCKER BEREIT (J/N)?";
4270 GET A$: IF A$="" THEN 4270
4280 IF A$<>"J" THEN 4270
4290 PRINT A$
4300 OPEN 4,4
4310 PRINT#4,"FILIALE NR. ";TE$(1);
4330 PRINT#4,CHR$(16);"20";"DATUM ";TE$(2);
4340 PRINT#4,CHR$(16);"40";"RECHNUNGS NR. ";TE$(3);
4345 PRINT#4,CHR$(16);"60";"BELEG NR. ";TE$(4)
4350 PRINT#4
4360 PRINT#4,"ARTIKEL NR.";
4375 PRINT#4,CHR$(16);"20";"WARENART";
4385 PRINT#4,CHR$(16);"60";"MWST GRUPPE"
4390 PRINT#4,"-----------";
4405 PRINT#4,CHR$(16);"20";"---------";
4415 PRINT#4,CHR$(16);"60";"-----------"
4420 FOR RN=1 TO 999
4430 GOSUB 10000
4440 GOSUB 10030
4442 IF VAL(TE$(1))<>RN THEN 4480
4444 IF LEFT$(TE$(2),1)="@" THEN 4480
4450 PRINT#4,TE$(1);
4460 PRINT#4,CHR$(16);"20";TE$(2);
4470 PRINT#4,CHR$(16);"60";TE$(3)
4480 NEXT RN
4490 PRINT#4: PRINT#4: PRINT#4
4999 CLOSE 4: CLOSE 8: CLOSE 15: GOTO 300
5000 OPEN 15,8,15: OPEN 8,8,8,"0:ARTDAT"
5002 GOSUB 12000
5005 PRINT CHR$(147);
5010 PRINT "**********************";
5020 PRINT "*ARTIKELVERWALTUNG VC*";
5030 PRINT "**********************";
5040 PRINT: PRINT
```

```
5050 PRINT "  DRUCKEN AUSWERTUNG": PRINT: PRINT
5060 FOR I=1 TO 4
5070 TE$(I)=""
5080 PRINT TT$(I)
5090 INPUT TE$(I)
5100 PRINT
5110 IF TE$(1)="ENDE" THEN 5999
5120 IF LEN(TE$(I))>TT(I) THEN 5070
5125 IF DV=0 AND I=2 THEN I=4
5130 NEXT I
5140 IF DV=1 THEN RETURN
5200 PRINT CHR$(147);
5210 PRINT "**********************";
5220 PRINT "*ARTIKELVERWALTUNG VC*";
5230 PRINT "**********************";
5240 PRINT: PRINT
5250 PRINT "  DRUCKEN AUSWERTUNG": PRINT: PRINT
5252 SA=0: SE=0: SG=0: SF=0: SH=0
5255 FOR I=1 TO 4
5256 M1(I)=0: M2(I)=0
5257 NEXT I
5260 PRINT "DRUCKER BEREIT (J/N)?";
5270 GET A$: IF A$="" THEN 5270
5280 IF A$<>"J" THEN 5270
5290 PRINT A$
5300 OPEN 4,4
5310 PRINT#4,"FILIALE NR. ";TE$(1);
5330 PRINT#4,CHR$(16);"20";"DATUM ";TE$(2);
5335 IF DV=0 THEN PRINT#4: GOTO 5350
5340 PRINT#4,CHR$(16);"40";"RECHNUNGS NR. ";TE$(3);
5345 PRINT#4,CHR$(16);"60";"BELEG NR. ";TE$(4)
5350 PRINT#4
5355 IF DV=1 THEN RETURN
5360 PRINT#4,"ART. NR.";
5370 PRINT#4,CHR$(16);"10";"ANZ.";
5375 PRINT#4,CHR$(16);"15";"WARENART";
5380 PRINT#4,CHR$(16);"40";"EINZ. EK";
```

```
5390 PRINT#4,CHR$(16);"50";"GESA. EK";
5400 PRINT#4,CHR$(16);"60";"EINZ. VK";
5410 PRINT#4,CHR$(16);"70";"GESA. VK"
5420 PRINT#4,"--------";
5425 PRINT#4,CHR$(16);"10";"----";
5430 PRINT#4,CHR$(16);"15";"--------";
5440 PRINT#4,CHR$(16);"40";"--------";
5450 PRINT#4,CHR$(16);"50";"--------";
5460 PRINT#4,CHR$(16);"60";"--------";
5470 PRINT#4,CHR$(16);"70";"--------"
5475 IF DV=1 THEN RETURN
5480 FOR RN=1 TO 799
5490 GOSUB 10000
5500 GOSUB 10030
5510 IF VAL(TE$(1))<>RN THEN 5580
5515 IF LEFT$(TE$(2),1)="@" THEN 5580
5520 PRINT#4,TE$(1);
5525 PRINT#4,CHR$(16);"10";TE$(4);
5530 PRINT#4,CHR$(16);"15";TE$(2);
5540 PRINT#4,CHR$(16);"40";TE$(5);
5550 PRINT#4,CHR$(16);"50";TE$(6);
5560 PRINT#4,CHR$(16);"60";TE$(7);
5570 PRINT#4,CHR$(16);"70";TE$(8)
5571 SA=SA+VAL(TE$(4))
5572 SE=SE+VAL(TE$(5))
5573 SG=SG+VAL(TE$(6))
5574 M1(VAL(TE$(3)))=M1(VAL(TE$(3)))+(VAL(TE$(6))*ME(VAL(TE$(3)))
     -VAL(TE$(6)))
5576 SF=SF+VAL(TE$(7))
5577 SH=SH+VAL(TE$(8))
5578 M2(VAL(TE$(3)))=M2(VAL(TE$(3)))+(VAL(TE$(8))-VAL(TE$(8))/ME(
     VAL(TE$(3))))
5579 IF DV=1 THEN RETURN
5580 NEXT RN
5590 PRINT#4,"--------";
5600 PRINT#4,CHR$(16);"10";"----";
5605 PRINT#4,CHR$(16);"15";"--------";
```

```
5610 PRINT#4,CHR$(16);"40";"--------";
5620 PRINT#4,CHR$(16);"50";"--------";
5630 PRINT#4,CHR$(16);"60";"--------";
5640 PRINT#4,CHR$(16);"70";"--------"
5650 PRINT#4,"SUMMEN:";
5660 PRINT#4,CHR$(16);"09";SA;
5670 PRINT#4,CHR$(16);"39";SE;
5680 PRINT#4,CHR$(16);"49";SG;
5690 PRINT#4,CHR$(16);"59";SF;
5700 PRINT#4,CHR$(16);"69";SH
5710 PRINT#4: PRINT#4: PRINT#4
5720 PRINT#4,"MWST. EK  6,5 % = ";M1(1);
5725 PRINT#4,CHR$(16);"40";"MWST. EK  7 % = ";M1(3)
5730 PRINT#4,"MWST. EK 13,0 % = ";M1(2);
5735 PRINT#4,CHR$(16);"40";"MWST. EK 14 % = ";M1(4)
5740 PRINT#4,"MWST. VK  6,5 % = ";M2(1);
5745 PRINT#4,CHR$(16);"40";"MWST. VK  7 % = ";M2(3)
5750 PRINT#4,"MWST. VK 13,0 % = ";M2(2);
5755 PRINT#4,CHR$(16);"40";"MWST. VK 14 % = ";M2(4)
5756 PRINT#4: PRINT#4: PRINT#4
5760 PRINT#4,"GESAMT EK. = ";SG+M1(1)+M1(2)+M1(3)+M1(4)
5770 PRINT#4,"GESAMT VK. = ";SH-M2(1)-M2(2)-M2(3)-M2(4)
5775 PRINT#4: PRINT#4: PRINT#4
5777 IF DV=1 THEN RETURN
5780 GOSUB 7000
5999 CLOSE 4: CLOSE 8: CLOSE 15: GOTO 300
6000 CLOSE 4: CLOSE 8: CLOSE 15
6010 PRINT CHR$(147);
6020 END
7000 S1=0: S2=0
7005 PRINT#4,"ART. NR.";
7010 PRINT#4,CHR$(16);"15";"ART";
7020 PRINT#4,CHR$(16);"40";"AUSGABEN";
7030 PRINT#4,CHR$(16);"60";"EINNAHMEN"
7040 PRINT#4,"--------";
7050 PRINT#4,CHR$(16);"15";"---";
7060 PRINT#4,CHR$(16);"40";"--------";
```

```
7070 PRINT#4,CHR$(16);"60";"----------"
7075 IF DV=1 THEN RETURN
7080 FOR RN=800 TO 999
7090 GOSUB 10000
7100 GOSUB 10030
7110 IF VAL(TE$(1))<>RN THEN 7180
7120 PRINT#4,TE$(1);
7130 PRINT#4,CHR$(16);"15";TE$(2);
7140 PRINT#4,CHR$(16);"40";TE$(5);
7150 PRINT#4,CHR$(16);"60";TE$(7)
7160 S1=S1+VAL(TE$(5))
7170 S2=S2+VAL(TE$(7))
7171 M2(VAL(TE$(3)))=M2(VAL(TE$(3)))+(VAL(TE$(7))-VAL(TE$(7))/ME(
     VAL(TE$(3))))
7172 M1(VAL(TE$(3)))=M1(VAL(TE$(3)))+(VAL(TE$(5))-VAL(TE$(5))/ME(
     VAL(TE$(3))))
7175 IF DV=1 THEN RETURN
7180 NEXT RN
7190 PRINT#4,"---------";
7200 PRINT#4,CHR$(16);"15";"---";
7210 PRINT#4,CHR$(16);"40";"---------";
7220 PRINT#4,CHR$(16);"60";"----------"
7230 PRINT#4,"SUMMEN:";
7240 PRINT#4,CHR$(16);"39";S1;
7250 PRINT#4,CHR$(16);"59";S2
7260 PRINT#4: PRINT#4: PRINT#4
7270 PRINT#4,"MWST. EK  6,5 % = ";M1(1);
7280 PRINT#4,CHR$(16);"40";"MWST. EK  7 % = ";M1(3)
7290 PRINT#4,"MWST. EK 13,0 % = ";M1(2);
7300 PRINT#4,CHR$(16);"40";"MWST. EK 14 % = ";M1(4)
7310 PRINT#4,"MWST. VK  6,5 % = ";M2(1);
7320 PRINT#4,CHR$(16);"40";"MWST. VK  7 % = ";M2(3)
7330 PRINT#4,"MWST. VK 13,0 % = ";M2(2);
7340 PRINT#4,CHR$(16);"40";"MWST. VK 14 % = ";M2(4)
7350 PRINT#4: PRINT#4: PRINT#4
7360 PRINT#4,"AUSGABEN = ";S1-M1(1)-M1(2)-M1(3)-M1(4)
7370 PRINT#4,"EINNAHMEN= ";S2-M2(1)-M2(2)-M2(3)-M2(4)
```

```
7380 PRINT#4: PRINT#4: PRINT#4
7999 RETURN
8000 PRINT CHR$(147);
8010 PRINT "**********************";
8020 PRINT "*ARTIKELVERWALTUNG VC*";
8030 PRINT "**********************";
8040 PRINT: PRINT
8050 PRINT " EINGABE KASSENZETTEL": PRINT: PRINT
8060 SA=0: SE=0: SG=0: SF=0: SH=0: S1=0: S2=0
8070 FOR I=1 TO 4
8072 M1(I)=0: M2(I)=0
8074 NEXT I
8080 PRINT "DRUCKER BEREIT (J/N)?";
8090 GET A$: IF A$="" THEN 8090
8100 IF A$<>"J" THEN 8090
8110 PRINT A$
8120 OPEN 4,4
8130 RETURN
8500 PRINT CHR$(147);
8510 PRINT "**********************";
8520 PRINT "*ARTIKELVERWALTUNG VC*";
8530 PRINT "**********************";
8540 PRINT: PRINT
8550 PRINT " EINGABE KASSENZETTEL": PRINT: PRINT
8560 GOSUB 5060: GOSUB 5310: RETURN
10000 HB=INT(RN/256): LB=RN-HB*256
10010 PRINT#15,"P"+CHR$(8)+CHR$(LB)+CHR$(HB)+CHR$(1)
10015 GOSUB 12000
10020 RETURN
10030 INPUT#8,TE$(1),TE$(2),TE$(3),TE$(4),TE$(5),TE$(6),TE$(7),TE$(8)
10060 RETURN
10070 TE$=TE$(1)+CHR$(13)+TE$(2)+CHR$(13)+TE$(3)+CHR$(13)+TE$(4)+CHR$(13

10072 TE$=TE$+TE$(5)+CHR$(13)+TE$(6)+CHR$(13)+TE$(7)+CHR$(13)+TE$(8)
10080 PRINT#8,TE$
10110 RETURN
12000 INPUT#15,X,X$,Y$,Z$
12010 IF X=0 OR X=50 THEN RETURN
```

```
12020 PRINT "FEHLER:";X,X$;Y$;Z$
12030 CLOSE 8
12035 CLOSE 15
12040 FOR I=1 TO 6000
12055 NEXT I
12060 GOTO 100
```

Dieses Programm verdeutlicht alle Teile, die in jeder Dateiverwaltung enthalten sind. Es wird sowohl mit einer gewissen Bildschirmmaske gearbeitet, als auch die die Benutzung der relativen Datei auf dem VC-20 und der Floppy 1540/1541 demonstriert. Da jeder Anwender andere Probleme hat, die mit derartigen Programmen gelöst werden können, haben wir an der Stelle nur ein konkretes Beispiel von vielen zeigen können. Der geübte Programmierer hat aber sicherlich keine Schwierigkeiten dieses Programm so umzuschreiben, daß es auch seine Aufgaben lösen kann.

Und wie geht's sequentiell ?

Wo Sie nun einen Eindruck von dem bekommen haben, wie man relative Dateien verarbeiten kann, sollte an dieser Stelle auch noch kurz auf die sequentielle Datei eingegangen werden.

Eine einfache Routine zum Schreiben von Daten auf Kassette könnte folgendermaßen aussehen:

```
 10 REM ****************************************
 20 REM SCHREIBEN VON NAME UND VORNAME AUF BAND
 30 REM VERSION FUER DATASETTE / VC-20
 40 REM ****************************************
 50 PRINT CHR$(147): REM BILDSCHIRM LOESCHEN
 52 PRINT "OEFFNEN DER DATEI"
 54 PRINT
 56 OPEN 1,1,1,"VC-20 DATEI"
 60 INPUT "NAME    : ";NA$
 70 INPUT "VORNAME : ";VN$
 80 PRINT
 90 PRINT "SCHREIBEN - NAME = ";NA$
100 PRINT "          - VORNAME = ";VN$
110 PRINT
120 PRINT#1,NA$
130 PRINT#1,VN$
140 PRINT
150 INPUT "WEITER (J/N) ";JN$
155 PRINT
160 IF JN$="J" THEN 60
170 IF JN$="N" THEN 200
180 PRINT "FALSCHE EINGABE !"
190 GOTO 140
200 CLOSE 1
210 END
```

Entsprechend lassen sich selbstverständlich auch Daten lesen und ändern.

7.2 Eine andere Methode Direktzugriff

Diese Art des Zugriffs auf Daten und Diskette wird leider oft völlig vernachlässigt. Zwar ist sie auch sehr kompliziert und aufwendig, hat aber auch einige sehr interessante Aspekte. Was läßt sich nun mit dem Direktzugriff alles verwirklichen ?

1.) Der Zugriff auf Dateien - Random-Dateien

Diese Methode hat einiges mit der sequentiellen Dateiverwaltung zu tun, ohne aber deren Nachteile zu haben, und einiges mit der relativen Dateiverwaltung.

2.) Der Zugriff auf einzelne Spuren der Diskette

Mit diesem Zugriff stehen Ihnen Möglichkeiten zur Verfügung, an die Sie bisher noch gar nicht dachten und deren Zweck Sie vielleicht noch gar nicht einsehen. Mehr dazu noch später.

1. Random-Dateien

Im Unterschied zu den sequentiellen und relativen Dateien, ist hier der einzelne Block immer 256 Bytes lang - auf einer Diskette gehen insgesamt 664 solcher Blöcke. Trotzdem können Sie auch kleinere Datensätze abspeichern. Es ist beispielsweise denkbar, daß Sie in einen Block insgesamt 4 Datensätze a' 64 Bytes Länge speichern. Sie müssen dann nur darauf achten, daß Sie innerhalb eines solchen Blocks auch genau auf den Satz zugreifen, den Sie tatsächlich verarbeiten wollen. Sie müssen zunächst eine sequentielle Datei öffnen, um dort die Spuren zu vermerken, in denen Sie die Datensätze gespeichert haben. Insgesamt benötigen Sie also 3 Dateien:

1.) sequentielle Datei für die Zeiger
2.) Kommando-Datei
3.) Daten-Datei zur direkten Speicherung

```
 10 OPEN 4,8,4,"CBM 64 DATEI,S,W": REM SEQUENT. DATEI
 20 OPEN 15,8,15: REM KOMMANDOKANAL
 30 OPEN 5,8,5,"#": REM DATEN-DATEI
 40 TE$="DATA BECKER GMBH"
 50 PRINT#5,TE$;",";1: REM TEXT, RECORD#
 60 T=1: S=1: REM TRACK=1, SECTOR=1
 70 PRINT#15,"B-A:";0,T,S: REM DRIVE, TRACK, SECTOR
 80 INPUT#15,ER,NA$,TR,BL: REM LESEN FEHLER
 90 IF ER=65 THEN T=TR: S=BL: GOTO 70
100 PRINT#15,"B-W:";5,0,T,S: REM SCHREIBE RECORD
110 PRINT#4,T;",";S
120 CLOSE 5
130 CLOSE 15
140 CLOSE 4
150 END
```

Was macht nun dieses Programm ? Zunächst werden die 3 notwendigen Dateien geöffnet, dann ein Text definiert, der direkt abgespeichert werden soll. Dieser Text wird in der Daten-Datei zunächst zwischengespeichert. Nun geht das Betriebssystem auf die Suche nach einem freien Block auf der Diskette. Die Suche beginnt bei Track 1, Sector 1, also dem Start der Diskette. Dazu wird die Zeile 70 benötigt. "B-A:" heißt BLOCK-ALLOCATE oder übersetzt soviel wie BLOCK-BELEGEN. Es wird also ein Block auf der Diskette gesucht, der noch nicht beschrieben wurde, bzw. wieder freigegeben wurde. Um zu sehen, ob der gesucht Block frei ist, muß der Kommandokanal gelesen werden. Hat er jedoch einen freien Block gefunden, so schreibt er den Text, der in der Daten-Datei bereit gestellt wurde, mit den gefundenen Blockadressen, auf die Diskette. Danach werden die Adressen noch in der sequentiellen Datei abgespeichert, damit der Satz später wiedergefunden wird. Gleichzeitig sperrt das Betriebssystem diese Spur automatisch, damit sie nicht unbeabsichtigt überschrieben wird. Die Dateien werden geschlossen und das Programm ist beendet.

Genauso interessant ist das Wiedereinlesen der Daten:

```
 10 OPEN 4,8,4,"CBM 64 DATEI"
 20 OPEN 15,8,15
 30 OPEN 5,8,5,"#"
 40 INPUT#4,T,S: REM LESEN DER ADRESSEN
 50 PRINT#15,"B-R:";5,0,T,S
 60 INPUT#5,TE$,RE
 70 PRINT#15,"B-F:";0,T,S
 80 CLOSE 5
 90 CLOSE 4
100 PRINT#15,"S:CBM 64 DATEI"
110 CLOSE 15
```

Und auch zu diesem Programm eine kurze Erläuterung: Nachdem die Dateien geöffnet wurden, wird die Adresse des Blocks eingelesen, in dem die eigentlichen Daten, in unserem Fall der Text, gespeichert sind. Nachdem dies geschehen ist, wird der Block durch den BLOCK-FREE Befehl wieder freigegeben. Dies darf natürlich nur dann geschehen, wenn der Block gelöscht werden kann. Danach werden die sequentielle- und die Daten-Datei wieder geschlossen, die sequentielle Datei wieder gelöscht und auch der Kommandokanal wieder geschlossen.

2. Der direkte Diskettenzugriff

Dieser Zugriff ermöglicht, wie sein Name schon sagt, den direkten Zugriff auf beliebige Tracks und Sectoren, also das Lesen von Informationen direkt von Disk, ohne erst Dateien zu eröffnen. So können Sie zum Beispiel leicht das Inhaltsverzeichnis lesen, ohne den LOAD "$",8 Befehl anwenden zu müssen und damit ein Programm, das im Speicher steht vielleicht zu zerstören. Oder Sie können ein Programm auf Diskette ändern, ohne es laden zu müssen - selbst zerstörte Programme können unter Umständen wieder repariert werden.

Diese Zugriffe sind aber so gefährlich, daß wir jeden Programmierer davor warnen möchten, sie ohne genaues Wissen über den Aufbau der Diskette, der Direktory, BAM oder des Programms selber anzuwenden. Es können so schnell ganze Dateien und Disketteninhalte verloren gehen. Lesen Sie dazu also auch genaustens das Handbuch zur Commodore Floppy 1541 durch. Sie werden verstehen, daß wir an dieser Stelle nicht auch noch ausführlich auf den ganzen Aufbau von Direktory, BAM etc. eingehen können. Das würde den Umfang dieses Buches sprengen. Wenn Sie aber gerne mit diesen Möglichkeiten experimentieren möchten, so empfehlen wir mit einer Testdiskette zu arbeiten - dann können zumindest keine "lebenswichtigen" Daten verloren gehen.

Zur Verdeutlichung der angewendeten Befehle hier noch eine Liste der Befehle, die direkt auf einen Block zugreifen können:

Bezeichnung:	Anwendung:
BLOCK-LESEN	"B-R:";KANALNUMMER,LAUFWERK,TRACK,BLOCK
BLOCK-SCHREIBEN	"B-W:";KANALNUMMER,LAUFWERK,TRACK,BLOCK
BLOCK-BELEGEN	"B-A:";LAUFWERK,TRACK,BLOCK
BLOCK-FREIGEBEN	"B-F:";LAUFWERK,TRACK,BLOCK
PUFFER-ZEIGER	"B-P:";KANALNUMMER,POSITION

Dieses sind die wichtigsten Befehle zum direkten Zugriff. Allen ist gemeinsam, daß Sie direkt auf den Floppy-Controller zugreifen. So haben Sie Möglichkeiten, die Sie bei einer gewöhnlichen Zugriffsart nie haben werden.

7.3 Programmoverlay ohne Variablenverlust

Bevor wir uns der Technik des Programmoverlays zuwenden, wollen wir erst einmal klären, was ein Overlay ist und welche Funktion es hat.

Bei größeren Programmen kommt man regelmäßig in die Verlegenheit, daß der Speicherplatz nicht ausreicht. Besonders bei intensiver Variablenbenutzung steigt das Programm sehr schnell mit einem 'OUT OF MEMORY ERROR' aus. Was soll man dagegen tun ? Der einzig gangbare Weg ist die Aufsplittung des Programms in mehrere Teile. Dies ist natürlich nur möglich, wenn das Programm aus mehreren voneinander unabhängigen Programmteilen besteht. Kommerzielle Programme, wie z.B. Dateiverwaltungen, arbeiten fast ausschließlich auf diese Art und Weise.
Die einzelnen Teile müssen dann bei Bedarf nachgeladen werden. Dies bedeutet natürlich einen gewissen Zeitverlust, der jedoch nicht besonders gegenüber der enormen Speicherplatzersparnis ins Gewicht fällt. Natürlich ist eine solche Technik vernünftig nur mit einem Floppylaufwerk realisierbar.
Damit haben wir auch schon geklärt, was ein Programmoverlay ist und wozu es dient. Fassen wir noch einmal zusammen: Bei einem Programmoverlay ist das eigentliche Programm in mehrere voneinander unabhängige Programme aufgespalten, die bei Bedarf von der Diskette nachgeladen werden.
Zum Glück besitzt der VC-20 alles in seinem BASIC, was zu einem Programmoverlay notwendig ist. Allerdings sind dabei ein paar Dinge zu beachten, wenn Variablen in den nachgeladenen Programmteil übergeben werden sollen. Gehen wir erst einmal von dem einfachen Fall aus, daß keine Variablen übergeben werden sollen.
In diesem Fall ist die Sache ganz einfach. Mit dem normalen LOAD-Befehl wird der zweite Programmteil nachgeladen:

```
1000 LOAD"TEIL2",8
```

Dies könnte eine Zeile aus dem ersten Programm sein. Gelangt das Programm in diese Zeile, wird "TEIL2" von der Diskette geladen und automatisch gestartet. Wäre nicht der kleine Zeitverlust, der Benutzer würde dies gar nicht bemerken.

Etwas schwieriger wird die Angelegenheit, wenn wir Variablen in den zweiten Teil übernehmen wollen. Grundsätzlich gilt, daß in diesem Fall das Programm auf keine Fälle länger sein darf als das im Speicher befindliche. Ist dies gewährleistet, so gehen keine Variablen verloren, da die Pointer auf das Ende des Programms nicht verändert werden. Diese Pointer sind gleichzeitig auch die Zeiger auf den Start der Variablen. Sollen mehrere Programmteile geladen werden, so muß von dem ersten Programm dieser Pointer entsprechend hochgesetzt werden. Beispiel : Das erste Programm hat auf der Diskette eine Länge von 10 Blöcken, das längste Programm hat eine Länge von 17 Blöcken. Zu Beginn des Programms muß nun die folgende Zeile stehen :

```
1 POKE46,PEEK(46)+8:CLR
```

Damit wird der Zeiger auf das Ende des Programms hochgesetzt. Die Zahl 8 gibt dabei die Differenz zwischen der Anzahl der Blöcke der beiden Programme +1 an.
Nun können Sie die einzelnen Programmteile laden, ohne Variablenverluste befürchten zu müssen. Achten Sie aber darauf, daß Zeile 1 nur einmal durchlaufen wird.
Ein kleines Problem ergibt sich nun aber noch bei Stringvariablen. Die Speicherung von Stringvariablen erfolgt normalerweise von der oberen Speichergrenze abwärts. Zu jeder Variablen existiert noch zusätzlich ein Pointer, der Länge der Variablen und Speicherstelle beinhaltet, an der der eigentliche Inhalt steht. Dieser Pointer befindet sich immer im normalen Variablenbereich. Wo aber nun der eigentliche Inhalt steht, hängt davon ab, wie Sie die Variable benutzt haben. Haben Sie die Variable durch eine Operation bzw. Zuweisung erhalten, so steht sie in dem normalen Stringspeicher und es ergeben sich keine Probleme für das Overlay. Eine solche Operation wäre zum

Beispiel:

```
10 A$ = A$ + RIGHT$(B$,2)
```

Ungünstiger sieht es aus, wenn Sie den Inhalt der Variablen direkt zuweisen oder aus DATA-Statements herausholen, zum Beispiel so:

```
10 A$="TEST"
```

oder

```
10 FORI=1TO2:READA$(I):NEXTI
20 DATA "TEST1","TEST2"
```

In diesem Fall werden die Inhalte der Variablen nicht noch einmal extra in dem normalen Stringspeicher gespeichert, sondern die Pointer zeigen direkt in den BASIC-Text. Dies ist aus ökonomischen Gründen sicher sinnvoll, hat jedoch für unser Overlay die Folge, daß wir den Inhalt der Variablen verlieren. Zwar werden beim Overlay die Pointer nicht verändert, allerdings ist das BASIC-PROGRAMM, in dem die Strings stehen, nach dem Laden des nächsten Programmes nicht mehr vorhanden.
Aus diesem Grunde müssen wir das Betriebssystem zwingen, die Variablen doch in dem normalen Stringspeicher abzulegen. Dies geschieht mit einem ganz einfachen Trick, wir addieren einen Leerstring:

```
10 A$="TEST"+""
```

oder

```
10 FORI=1TO2:READA$:A$=A$+"":NEXTI
20 DATA "TEST1","TEST2"
```

Nun zeigen die Pointer für die Strings nicht mehr in das BASIC-Programm. Ihrem Overlay steht jetzt nichts mehr im Wege!

7.4 Textverarbeitung auf dem VC-20

Seit es Computer gibt, gibt es immer das Verlangen, Arbeiten, die bisher mühsam von Hand erledigt wurden, nun mit Hilfe dieser elektronischen Wunderwerke zu bewältigen. Eines der aufwendigsten Probleme ist die Erstellung von Texten, Berichten, und alles das, was bisher auf einer Schreibmaschine unter großem Zeit- und Arbeitsaufwand geschrieben wurde. Auch der VC-20 gehört zu den Computern, die es erlauben, Texte am Bildschirm zu erstellen, zu ändern, abzuspeichern und auf einem Drucker auszugeben. Solche Programme gibt es zu hunderten - sie werden heute schon in sehr vielen Fachzeitschriften ausgedruckt. Aber was hinter der Idee der Textverarbeitung steht, das erklärt einem niemand genau. Wir wollen das an dieser Stelle ändern.

In diesem Kapitel erhalten Sie einen Einblick hinter die Kulissen der Textverarbeitung. Es wird zwar kein komplettes Textprogramm vorgestellt, dafür erhalten Sie aber Informationen, mit denen es Ihnen nicht mehr ganz so schwer fallen dürfte, mit etwas übung und Geduld ein eigenes, auf Ihre Belange zugeschnittenes Textprogramm zu erstellen. Denn dies ist oftmals das eigentliche Problem: Jeder möchte seine Texte in einem ganz speziellen Format erstellen. Der eine Anwender benötigt unbedingt eine Rechenfunktion während der andere noch eine Dateiverwaltung dazu wünscht und vieles mehr. Ein Programm solchen Ausmaßes würde den Umfang eines solchen Buches sprengen (und hätte schon nichts mehr mit TIPS und TRICKS zu tun). Aus diesem Grund wollen wir an dieser Stelle nur die wichtigsten Funktionen eines Textprogrammes vorstellen.

Als erstes wollen wir uns hier mit der Eingabe von verschiedenen Zeichen befassen. Dabei müssen wir uns zunächst für eine der beiden Möglichkeiten entscheiden, die das VC-20 BASIC bietet: INPUT oder GET. Falls wir den INPUT-Befehl verwenden wollen, was sicherlich die einfachste Möglichkeit ist, müssen wir uns über ein paar Probleme im klaren sein. Das größte Problem ist, daß wir die Tastatur und damit die Eingaben nicht richtig kontrollieren

können. Das heißt, daß alle Zeichen, die wir über die Tastatur aus eingeben, tatsächlich akzeptiert werden. Nun kann es aber passieren, daß ein Text länger als die erlaubten 88 Zeichen wird (per Zeile versteht sich) und der VC-20 bei der Betätigung der RETURN-Taste einen Fehler meldet. Oder aber wir haben noch gar kein Zeichen eingegeben und betätigen aus irgend einem Grund die DEL-Taste. Auch dies kann zu einem Fehler führen. - Wir sehen also, daß diese Form der Eingabe für eine komfortable Texteingabe nicht in Frage kommen kann.

Bei der Textverarbeitung kommt es darauf an, jede Eingabe über Tastatur kontrollieren zu können (Ist die Zeile nicht zu lang? Darf diese Taste überhaupt gedrückt werden? usw.). Aus diesem Grund verwenden wir den GET-Befehl. Ein Programm zur Eingabe eines Textes von maximal 99 Zeilen könnte so aussehen:

```
100 PRINT CHR$(147);
110 DIM TE$(100)
120 INPUT "ZEILENNUMMER: ";ZE
130 PRINT
140 IF ZE<1 OR ZE>99 THEN 120
160 PRINT ZE;": ";
170 AZ=0
180 GET BS$
190 IF BS$="" THEN 180
200 IF BS$<>CHR$(20) THEN 210
202 IF LEN(TE$(ZE))<1 THEN 180
204 PRINT CHR$(157);" ";CHR$(157);
206 TE$(ZE)=LEFT$(TE$(ZE),LEN(TE$(ZE))-1)
208 GOTO 180
210 IF BS$=CHR$(17) THEN 280
230 IF AZ=0 THEN TE$(ZE)="": AZ=1
240 PRINT BS$;
250 IF BS$=CHR$(13) THEN ZE=ZE+1: GOTO 272
260 TE$(ZE)=TE$(ZE)+BS$
270 IF LEN(TE$(ZE))=80 THEN ZE=ZE+1
272 IF ZE=100 THEN 280
```

```
275 GOTO 180
280 TE$(ZE)="!!"
```

ACHTUNG: Wenn Sie dieses Programm laufen lassen, sehen Sie keinen Cursor mehr. Dies ist eine Besonderheit des GET-Befehls und läßt sich durch eine kleine Änderung im Programm teilweise beheben. Sie müssen nur an Stelle des Cursors ein anderes Zeichen voranschicken. (z.B. CHR$(185)). Das Programm muß dann aber etwas geändert werden:

```
185 PRINT CHR$(185);CHR$(157);
203 PRINT " ";CHR$(157);
240 PRINT " ";CHR$(157);BS$;
280 PRINT " ";CHR$(157)
290 TE$(ZE)="!!"
```

Mit diesem Programm können Sie nun, beginnend bei einer bestimmten Zeilennummer, Ihren Text eingeben. BS$m steht für "Buchstabe" und enthält das Zeichen, das wir über die Tastatur aus eingegeben haben. Zunächst wird abgefragt, ob wir die DEL-Taste gedrückt haben. Wenn noch kein anderes Zeichen vorher eingegeben wurde, geschieht nichts - man kehrt wieder zur Eingabe zurück. Ansonsten wird das Zeichen leer überschrieben und innerhalb des Satzes gelöscht.

Wenn Sie keinen Text mehr eingeben wollen, drücken Sie einfach die Pfeil-abwärts-Taste. Das Programm bricht dann an dieser Stelle ab.

Nun wollen wir aber noch den gerade eingegebenen Text auf dem Drucker wieder ausgeben. Hierbei wollen wir nun noch unterscheiden, ob die Zeilennummern mit angegeben werden sollen oder nicht.

```
300 PRINT CHR$(147);
310 INPUT "MIT ZEILENNUMMERN ";JN$
320 IF JN$="J" THEN ZD=1: GOTO 350
```

```
330 IF JN$="N" THEN ZD=0: GOTO 350
340 GOTO 310
350 OPEN 4,4
360 FOR ZE=1 TO 100
370 IF TE$(ZE)="!!" THEN 385
380 IF ZD=1 THEN PRINT#4,ZD;": ";
382 PRINT#4,TE$(ZE)
384 NEXT
385 PRINT#4
390 CLOSE 4
```

Dieses sehr einfache Druckprogramm gibt den Text auf dem Drucker aus, der mit dem Eingabeprogramm in den Variablen TE$(1) bis TE$(n) (n = Anzahl der Zeilen) eingegeben wurde. Damit nicht alle 100 Zeilen gedruckt werden, falls weniger eingegeben wurden, druckt das Programm nur solange, bis es auf den Text "!!" in einer Zeile stößt. Das bedeutet, daß hier der Text zu Ende ist. Dieses Zeichen wurde ja vom Eingabeprogramm selber an den Schluß des Textes angefügt. Um nach der Eingabe eines Textes diesen auch gleich drucken zu können, geben Sie nun noch folgende Zeilen ein:

```
292 INPUT "TEXT DRUCKEN (J/N) ";JN$
295 IF JN$="N" THEN END
```

Sie wollen aber sicherlich nicht jedesmal einen Text neu eingeben um ihn dann auszudrucken - das wäre ja überhaupt keine Erleichterung gegenüber der Eingabe auf Schreibmaschine. In diesem Fall wollen wir den Text auf eine Kassette speichern. Hier können wir nun sogar das sequentielle Abspeichern wählen (relative Abspeicherung wäre sowieso nur auf einer Floppy möglich), da ein Text immer sequentiell abgearbeitet werden muß. Hier nun die kleine Speicherroutine:

```
400 OPEN 1,1,1,"TEXTDATEI"
410 FOR ZE=1 TO 100
420 IF TE$(ZE)="!!" THEN 460
430 PRINT#1,TE$(ZE)
```

```
440 NEXT ZE
460 CLOSE 1
470 END
```

Damit auch diese Routine vom Programm aus aufgerufen werden kann, müssen wir noch ein paar Zeilen einfügen und ändern:

```
295 IF JN$="N" THEN 396
396 INPUT "TEXT SPEICHERN (J/N) ";JN$
397 IF JN$="N" THEN END
```

Nach dem selben Prinzip arbeitet auch die Routine zum Einlesen eines Textes von Kassette. Die Datei muß hierbei nur zum Lesen geöffnet werden (s. Kapitel 5.9) und die einzelnen Zeilen müssen mit dem INPUT#-Befehl gelesen werden.

Diese wenigen Routinen lassen schon deutlich das Gerüst einer Textverarbeitung erkennen. Die Routinen EINGABE, AUSGABE, SPEICHERN und LESEN sind Hauptbestandteil jeden Textprogrammes. Diese Routinen werden dann noch durch verschiedene Besonderheiten erweitert. Es gibt zum Beispiel bei fast jedem Textprogramm die Möglichkeit nach verschiedenen Wörtern oder Zeichen innerhalb des Textes zu suchen. Eine Routine, die diese Aufgabe erfüllt finden Sie in diesem Buch. Außerdem bietet ja auch das BASIC des VC-20 schon ein paar sehr leistungsfähige Befehle zur Textein- und Ausgabe. So lassen sich zum Beispiel durch Befehle wie LEFT$, RIGHT$ und MID$ ganze Zeilen in verscheidene Teile zerlegen.

Es ist also auch mit einfachen Mitteln schon möglich eine kleine Textverarbeitung zu schreiben. Noch ein Tip am Rande. Gerade bei solchen Programmen ist es sehr sinnvoll die verschiedenen Routinen in Maschinensprache zu schreiben (s. die Routine zum Suchen eines Textteils innerhalb eines Textes).

7.5 Suchspiel - Goldgräber

Da Sie mit dem VC-20 einen Computer besitzen, der hevorragende Grafik- und Soundmöglichkeiten besitzt, können Sie natürlich auch sehr gut Spiele realisieren. In diesem Kapitel wollen wir uns aus der großen Palette der Spiele zwei Gruppen herausgreifen, die beide besonders von Grafik und Sound unterstützt werden können. Diese beiden Gruppen sind Such- und Geschicklichkeitsspiele. Zu beiden Kategorien werden Sie je ein komplettes Programm bekommen, ausführlich dokumentiert, damit Sie es verstehen können und nach Ihren eigenen Wünschen verändern können. Wenden wir uns zuerst dem Suchspiel zu:

In unserem Suchspiel müssen Sie in einem Bergwerk Gold suchen. Dazu wählen Sie zuerst die Etage, von der Sie starten (A-S) und dann können Sie sich mit Hilfe der Tasten I,J,K und M durch das Bergwerk bewegen. Doch Sie können nicht nur auf Gold stoßen, sondern auch einen Wassereinbruch verursachen, auf Granit stoßen oder in Fallgruben stürzen. Jeder Schritt im Bergwerk kostet Sie 10 Dollar, ein Wassereinbruch 2/3, eine Fallgrube die Hälfte Ihres Kapitals und Granit mit der notwendigen Sprengung 250 Dollar. Stoßen Sie auf eines der fünf Goldstücke, so wird Ihr Kapital je nach Größe des Goldstückes erhöht und dient Ihnen zur Finanzierung der weiteren Suche. Das Spiel ist zu Ende, wenn Sie entweder alle 5 Goldstücke gefunden haben oder Ihr Kapital aufgebraucht ist.

Anhand dieser umfangreichen Beschreibung merken Sie sicher schon, daß dies alles nicht ganz leicht zu programmieren ist. Doch keine Angst, wir gehen das Programm jetzt systematisch durch. Sie finden auf den nächsten Seiten das Listing, dann die Erläuterungen dazu.

```
1 POKE650,128
2 IFPEEK(36869)=240THENVI=7680:FA=38400
3 IFPEEK(36869)=192THENVI=4096:FA=37888
10 PRINT"▒
41 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒GOLD RAIDERS"
45 PRINT"▒▒▒▒▒KANN'S LOSGEHEN(J/N)?"
170 GETA$:IFA$=""THEN189
189 IFA$<>"J"THEN 170
190 PO=PO-B-22
200 SC=10000:PRA=250
1030 POKE 36879,12
1036 POKE 36878,15
1037 ERZ=1
1038 HOE=0
1040 PRINT"▒
1100 FOR X=20TO21
1110 POKEFA+484-22*A+X,1
1111 POKE 36875,128+X
1115 POKEVI+484-22*A+X,160
1120 NEXTX
1121 POKE 36875,0
1130 A=A+1:IF A<19 THEN 1100
1200 FOR X=0TO3
1205 POKE 36875,128+3*X
1210 POKEVI+66-22*X+S,160:POKEFA+66-22*X+S,1
1220 NEXTX
1221 POKE 36875,0
1230 S=S+1:IFS<22THEN1200
1700 FORETAGE=1TO19
1710 POKEVI+85+22*ETAGE,ETAGE
1715 POKEFA+85+22*ETAGE,1
1720 NEXTETAGE
2022 GEGE=102
2030 A=0
2035 FORVER=0TO16
2040 GOSUB 10000
2050 IFPEEK(VI+484-22*Y1+X1)<>32THEN 2040
```

```
2060 POKEFA+484-22*Y1+X1,0
2065 POKEVI+484-22*Y1+X1,GEGE
2070 A=A+1
2075 IFA>=0THENGEGE=102:IFA>3THENGEGE=7 :IFA>8THENGEGE=78:IFA>12THENGEGE=105
2080 NEXTVER
2090 GOTO 20000
10000 Y1=INT(RND(1)*19)+0
10010 X1=INT(RND(1)*19)+0
10020 RETURN
20000 POKEVI+81,160
20050 POKEFA+81,0
20225 FOR VER=0TO7
20235 FOR VER=0TO7
20500 GETA$:IFA$=""THEN20500
20505 BUCH=ASC(A$)
20506 IF BUCH<65 THEN 20500
20507 IF BUCH>83 THEN 20500
20510 FOR L=0TO6
25000 IFL>1THENPOKEVI+81+L-1,160:POKEFA+81+L-1,1
25010 POKEVI+81+L,160:POKEFA+81+L,3
25020 NEXTL
30100 FOR AUFZ=0TOBUCH-65
30105 POKEVI+65+22*AUFZ,160:POKEFA+87+22*AUFZ,1
30106 POKE 36874,128
30107 FORTGH=0TO30:NEXT
30110 POKEVI+87+22*AUFZ,160
30112 POKEFA+109+22*AUFZ,3
30120 NEXTAUFZ
30121 POKE 36874,0
40000 REM
40030 POKEVI+64+22*AUFZ,160:POKEFA+86+22*AUFZ,3
40500 POKEVI+85+22*AUFZ,170:POKEFA+85+22*AUFZ,3
40510 PO=VI+84+22*AUFZ
40520 PF=FA+84+22*AUFZ
40530 B=1:C=0
40600 GETA$:IFA$=""THEN40600
40610 IFA$="J"THENB=B-1:GOTO40900
40620 IFA$="K"THENB=B+1:GOTO40900
```

```
40630 IFA$="I"THENC=C-1:GOTO40900
40640 IFA$="M"THENC=C+1:GOTO40900
40660 GOTO 40600
40670 REM
40900 IF B<-18THENB=B+1:GOTO40600
40910 IF B>0THENB=B-1:GOTO40600
40920 IF C+AUFZ<1THENC=C+1:GOTO 40600
40930 IF C+AUFZ>19THENC=C-1:GOTO 40600
42005 IFPEEK(PO+B+22*C)=105 THEN M=105:GOTO 53000
42011 IFPEEK(PO+B+22*C)=7  THENM=7:GOTO 53000
42012 IFPEEK(PO+B+22*C)=102THENM=102:GOTO 53000
42013 IFPEEK(PO+B+22*C)=78 THENM=78:GOTO 53000
52000 POKE PO+B+(22*C),170
52001 POKE PF+B+22*C,1
52002 FOR U=128TO240STEP3
52003 POKE 36877,U:NEXTU
52006 POKE 36877,0
52009 POKE PO+B+22*C,160
52010 POKE PF+B+(22*C),1
52011 PRA=PRA+INT(RND(1)*20)+-5:SC=SC-10
52013 FGH=INT(SC)
52014 IFFGH<10000 THEN POKEVI+16,160:POKE FA+16,1
52015 IFFGH<1000  THEN POKEVI+15,160:POKEFA+15,1
52016 IFFGH<100   THEN POKEVI+14,160:POKEFA+14,1
52017 IFFGH<10 THENSC=0:GOTO63000
52019 PRINT"[illegible]BANKKONTO ="INT(SC)
52020 IFGDH=5 THEN 63000
52030 GOTO 40600
53000 POKE PO+B+22*C,M:POKEPF+B+22*C,1
53010 IF M=7THEN 60000
53020 IF M=102 THEN61000
53030 IF M=105 THEN 62500
53040 IF M=78 THEN 61500
60000 REM
60001 GDH=GDH+1
60010 FOR JK=0TO200
60015 UNZ=INT(RND(1)*6)+3
60020 POKE 36875,INT(RND(1)*128)+128:NEXTJK
60030 POKE 36875,0
60040 POKE PO+B+22*C,160:POKE PF+B+22*C,1
```

```
60050 SC=SC+UNZ*PRA
60060 GOTO 52000
60900 REM
61000 FOR WA=0TO19-AUFZ-C
61010 FOR WE=0TO18
61011 PED=PEEK(FA+484+WE-22*WA)AND15
61014 IF PED=1THEN61020
61016 GOTO 61030
61020 POKEFA+484+WE-22*WA,6
61030 POKE 36877,128+4*WA
61040 NEXT WE
61060 NEXT WA
61061 POKE PO+B+22*C,160:POKEPF+B+22*C,6
61070 POKE 36877,0
61080 SC=SC/3:GOTO52000
61500 REM
61520 FOR DF=200 TO140 STEP-1
61525 FORTHN=0TO15:NEXT
61530 POKE 36876,DF:NEXTDF
61540 SC=SC/2
61545 IF C+AUFZ<19 THEN POKE PO+B+22*C,160:POKEPF+B+22*C,1
61546 IF C+AUFZ=19 THEN POKE PO+B+22*C,160:POKEPF+B+22*C,1:POKE 36876,0:GOTO 520
00
61549 POKE 36876,0
61550 C=C+1:GOTO 52000
62000 GOTO 52000
62500 REM
62505 KH=1
62510 FOR HN=128TO240STEPKH
62520 POKE 36877,HN:NEXTHN
62521 KH=KH+1
62525 IF KH<9THEN62510
62530 POKE PO+B+22*C,160:POKEPF+B+22*C,1
62540 SC=SC-250
62550 GOTO 52000
63000 PRINT"
63010 POKE 36879,25
63020 PRINT"SPIELENDE"
63030 PRINT"SIE HABEN AUF IHREM   KONTO <"INT(SC)">DOLLAR."
```

Damit das Programm auf allen Ausbaustufen Ihres VC-20 läuft, wird in den Zeilen 2 und 3 bestimmt, wo Video- und FarbRAM liegen. Bis Zeile 2090 wird das Spiel initialisiert. Dabei wird zuerst bis Zeile 2022 der Bildschirm initialsiert und bis Zeile 2090 werden Gold, Granit, Wassereinbrüche und Fallgruben versteckt. In Zeile 20500 wird dann die Startetage abgefragt und von Zeile 20510 bis 30120 gesetzt. In Zeile 40600 wird die Bewegung abgefragt. In den Zeilen 40610 bis 40660 wird auf eine gültige Abfrage geprüft. In den Zeilen 40900 bis 40930 wird überprüft, ob Sie das Bergwerk auch nicht verlassen. Der Test auf einen Fund findet in den Zeilen 42005 bis 42013 statt. Hat kein Fund stattgefunden, wird die neue Position in den Zeilen 52000 bis 52010 gesetzt. Anschließend wird Ihr Kapital vermindert und ausgedruckt. Wenn Sie noch nicht alle fünf Goldstücke gefunden haben (Zeile 52020),wird in Zeile 52030 wieder zurückverzweigt und ein neuer Abfragezyklus beginnt. In den Zeilen ab 53000 werden die vier möglichen Funde behandelt. Zuerst wird in den Zeilen 60000 bis 60060 der Goldfund ausgeführt. Dabei werden die Geräusche ausgeben und der Kapitalbetrag erhöht. In den Zeilen 60900 bis 61080 wird der Wassereinbruch behandelt. Die Zeilen 61500 bis 62000 bearbeiten die Auswirkungen eines Sturzes in eine Fallgrube, die Zeilen 62500 bis 62550 bearbeiten die Ausführung einer Sprengung. In den letzten Zeilen ab 63000 schließlich wird die Meldung über das Spielende ausgegben und der gewonnene Betrag ausgegeben.

7.6 Geschicklichkeitsspiel - STARSHOOTER

In dem Spiel dieses Kapitels wird kaum Strategie verlangt. Wichtig ist hier Ihre Geschicklichkeit im Umgang mit der Tastatur. Worum geht es nun? Auf dem Bildschirm erscheinen 20 zufällig verteilte Sterne. Ihre Aufgabe ist es nun, mit Hilfe eines Raumschiffes die Sterne abzuschießen. Das Problem ist aber, daß Sie Ihr Schiff nicht stoppen können und daß immer wieder neue Sterne entstehen. Das Spiel ist zuende, wenn der Bildschirm sauber ist, d.h. alle Sterne sind verschwunden oder Sie mit einem der Sterne kollidieren sollten. Wie sieht nun der Aufbau eines solchen Spieles grundsätzlich aus?
Wir unterscheiden wieder zwischen zwei Hauptteilen, der Initialisierung und dem eigentlichen Spielteil. Bei der Initialisierung haben wir vier Aufgaben:

1) Verlegen des Charactergenerators und Erzeugen der Spielfigur (Raumschiff)
2) Initialisieren des Bildschirms (Setzen der Sterne und der Farben)
3) Erzeugen der zufälligen Startposition
4) Erzeugen der zufälligen Bewegungsrichtung

Schauen Sie sich das Listing auf der übernächsten Seite an. In Zeile 1 setzen wir den für BASIC verfügbaren Speicherplatz herunter und kopieren einen Teil des Charactergenerators in den geschützten Bereich. In Zeile 20 wird das Raumschiff erzeugt. Da wir vier verschiedene Bewegungsrichtungen haben und unser Raumschiff nicht rückwärts fliegen soll, schaffen wir vier Raumschiffe, die sich nur durch die Lage auf dem Bildschirm unterscheiden. Dabei gilt folgendes für die Bewegung:

1 : Das Raumschiff fliegt nach rechts
2 : Das Raumschiff fliegt nach unten
3 : Das Raumschiff fliegt nach links
4 : Das Raumschiff fliegt nach oben

In Zeile 40 beginnt die Initialisierung des Bildschirms. Der Bildschirm wird gelöscht und das FarbRAM initialisiert. Ab Zeile 60 werden die Sterne erzeugt. Die Abfrage in Zeile 90 bewirkt, daß genau 20 Sterne erzeugt werden. In Zeile 100 werden die zufällige Startposition und in Zeile 110 die Bewegungsrichtung festgelegt. Dann wird das Raumschiff auf den Bildschirm gebracht. Beachten Sie, daß die Variable R die Richtung anzeigt und gleichzeitig das richtigliegende Raumschiff auswählt.
Wenden wir uns nun dem eigentlichen Spielteil zu. Was haben wir zu tun?

1) Kollision abfragen
2) Bewegung ausführen
3) Tastatur abfragen
4) Schuß abfragen
5) Spielende prüfen
6) neue Hindernisse erzeugen

Diese grobe Gliederung finden Sie in den Zeilen 200 bis 260 wieder. Schauen wir uns die dazu notwendigen Unterprogramme an:
zu 1) Zuerst wird die alte Position des Raumschiffes gelöscht und dann die neue Position errechnet(1110-1130). Dann wird in Zeile 1710 geprüft, ob an dieser Position ein Stern sitzt. Ist dies nicht der Fall, wird das Programm mit Punkt 2 fortgesetzt. Ist an der errechneten Stelle ein Stern vorhanden, so wird zu Zeile 2100 verzweigt. Dort wird eine Abschlußmeldung ausgegeben und das Programm beendet.

zu 2) Die Bewegung selber ist so schnell erledigt, daß wir dafür kein extra Unterprogramm benötigen. Die alte Position haben wir bereits unter Programmpunkt 1 gelöscht und ebenfalls dort die neue Position errechnet. Nun brauchen wir nur noch das Raumschiff

wieder auf den Bildschirm zu bringen (Zeile 210). Dazu geben wir noch ein Geräusch aus.

zu 3) Auch die Tastaturabfrage ist schnell abgehandelt. In den Zeilen 1200 bis 1250 werden die 5 erlaubten Tasten abgefragt und entsprechend die Variablen gesetzt.

zu 4) In Zeile 1310 wird überprüft, ob überhaupt ein Schuß abgefeuert wurde. Da nur in Flugrichtung geschossen werden kann, wird nun entsprechend R zu einem von vier identisch aufgebauten Programmteilen verzweigt. Wir behandeln hier deshalb exemplarisch nur die Zeilen 1300 bis 1390. In Zeile 1325 wird die Position des Rausmschiffes in eine andere Variable gerettet. Wir verwenden hier wieder die Variablen X und P, um das Unterprogramm in Zeile 1010 verwenden zu können. Wir bauen nun in Zeile 1330 eine Schleife auf. Der Schuß soll von der Position des Raumschiffes beginnend bis zum Rand des Bildschirms gehen. Im Unterprogramm ab Zeile 1010 berechnen wir die Position des Schusses und prüfen anschließend, ob auf dieser Position kein Stern vorhanden ist. Ist dies der Fall, so wird in Zeile 1390 zuerst ein Schußgeräusch ausgeben (GOSUB1900) und dann die Schleife weitergeführt. Ist der Schuß am Rand angelangt, so werden alle Variablen wieder zurückgesetzt und das Unterprogramm beendet. Ist ein Stern vorhanden, so wird in Zeile 1340 dieser Stern gelöscht, anschließend ein Treffergeräusch ausgeben, die Variablen wieder zurückgesetzt und das Unterprogramm beendet. Diese Technik bewirkt, daß mit einem Schuß immer nur ein Stern gelöscht werden kann. Wichtig ist hier auch noch der Zähler der vorhandenen Sterne (H). Ist H=0, dann hat man alle Sterne abgeschossen und das Spiel ist beendet.

zu 5) Wie bereits erwähnt, existieren zwei Möglichkeiten, das Spiel zu beenden. Entweder kollidiert das Raumschiff mit einem Stern (das Spiel ist verloren) oder man schießt alle Sterne ab. Den ersten Fall haben wir bereits unter dem Punkt abgehandelt, bliebe also noch der zweite Fall. Die Anzahl der noch vorhandenen Sterne steht in der Variablen H, so daß wir diese nur auf den Wert Null zu überprüfen brauchen (Zeile 240). Ist H=0, so wird in Zeile 2200 verzweigt, wo eine entsprechende Meldung mit der dazugehörigen Geräuschkulisse ausgegeben wird.

zu 6) Die Position der neuen Sterne wird in der Zeile 250

berechnet. Wie Sie sehen, sind Werte zwischen 0 und 100 für die X und Y-Richtung möglich. Dies hat zur Folge, das nicht jeder Versuch, einen Stern zu erzeugen, Erfolg hat. Die Schwierigkeit des Spieles läßt sich also sehr gut an dieser Stelle steuern. Erhöhen Sie den Wert in der Berechnung, so werden weniger Sterne erzeugt, vermindern Sie den Wert, erscheinen immer öfter neue Sterne auf dem Bildschirm. In Zeile 260 wird schließlich überprüft, ob die Position im erlaubten Rahmen liegt und entsprechend ein neuer Stern auf dem Bildschirm dargestellt.

Wir haben Ihnen nun bis ins kleinste Detail den Aufbau eines solchen Spieles erklärt. Wichtig ist, daß Sie sich, bevor Sie sich an den Rechner setzen, genau überlegen, wie Sie das Programm aufbauen und aus welchen Teilen es besteht. Sie haben gesehen, daß Sie so ein klar gegliedertes Programm erstellen können, daß übersichtlich ist und keine Fehler enthält. Das nachfolgende Programm läßt Ihnen noch viele Freiheiten für eigene Erweiterungen. Lassen Sie Ihre Fantasie spielen und überlegen Sie sich, was noch zu verbessern wäre. Sicher finden Sie noch eine Fülle von Möglichkeiten. Wir wünschen Ihnen jedenfalls viel Spaß beim Programmieren und Spielen.

```
5 REM * * *  S T A R S H O O T E R  * * *
10 POKE56,28:POKE52,28:CLR
15 FORI=7168TO7679:POKEI,PEEK(25600+I):NEXTI
20 FORI=7176TO7207:READA:POKEI,A:NEXTI
30 POKE36869,255:POKE36879,8:POKE36878,15
40 PRINTCHR$(147)
50 FORI=38400TO38905:POKEI,2:NEXT
60 GOSUB1000:GOSUB1010
80 IFPEEK(P)<>32THEN60
90 POKEP,42:H=H+1:IFH<20THEN60
100 GOSUB1000:GOSUB1010:IFPEEK(P)<>32THEN100
110 R=INT(RND(1)*3)+1:POKEP,R
200 GOSUB1700:REM KOLLISION?
210 POKEP,R:POKE36877,220:POKE36877,200:POKE36877,240:REM BEWEGUNG
220 GOSUB1200:REM ABFRAGE TASTATUR
230 GOSUB1300:REM SCHUSSABFRAGE
240 IFH=0THEN2200:REM ALLE HINDERNISSE WEG
250 XZ=RND(1)*100-1:YZ=RND(1)*100-1:PZ=7680+XZ+YZ*22
260 IFXZ<22ANDYZ<23ANDPEEK(PZ)=32THENPOKEPZ,42:H=H+1
998 GOTO200
999 END
1000 X=INT(RND(1)*21):Y=INT(RND(1)*22):RETURN
1010 P=7680+X+Y*22
1020 RETURN
1100 POKEP,32
1110 X=X-(R=3)*(X>0)+(R=1)*(X<21)
1120 Y=Y-(R=4)*(Y>0)+(R=2)*(Y<22)
1130 GOSUB1010:RETURN
1200 GETA$
1210 IFA$="I"THENR=4
1220 IFA$="J"THENR=3
1230 IFA$="M"THENR=2
1240 IFA$="K"THENR=1
1250 IFA$=" "THENS=1
1260 RETURN
1300 IFS=0THENRETURN
1320 IFR>1THEN1420
1325 XZ=X:PZ=P
1330 FORX=XZ+1TO21:GOSUB1010:IFPEEK(P)=32THEN1390
1340 POKEP,32:GOSUB1950:X=XZ:P=PZ:S=0:H=H-1:RETURN
1390 GOSUB1900:NEXTX:X=XZ:S=0:P=PZ:RETURN
```

```
1420 IFR>2THEN1520
1425 YZ=Y:PZ=P
1430 FORY=YZ+1TO22:GOSUB1010:IFPEEK(P)=32THEN1490
1440 POKEP,32:GOSUB1950:Y=YZ:P=PZ:S=0:H=H-1:RETURN
1490 GOSUB1900:NEXTY:Y=YZ:S=0:P=PZ:RETURN
1520 IFR>3THEN1625
1525 XZ=X:PZ=P
1530 FORX=XZ-1TO0STEP-1:GOSUB1010:IFPEEK(P)=32THEN1590
1540 POKEP,32:GOSUB1950:X=XZ:P=PZ:S=0:H=H-1:RETURN
1590 GOSUB1900:NEXTX:X=XZ:S=0:P=PZ:RETURN
1625 YZ=Y:PZ=P
1630 FORY=YZ-1TO0STEP-1:GOSUB1010:IFPEEK(P)=32THEN1690
1640 POKEP,32:GOSUB1950:Y=YZ:P=PZ:S=0:H=H-1:RETURN
1690 GOSUB1900:NEXTY:Y=YZ:S=0:P=PZ:RETURN
1700 GOSUB1100
1710 IFPEEK(P)=42THEN2100:REM KOLLISION
1720 RETURN
1900 POKE36877,230:POKE36877,235:RETURN
1950 POKE36877,0:POKE36875,240:FORI=1TO9:NEXTI:POKE36875,0:RETURN
2000 DATA 0,224,248,143,248,224,0,0
2010 DATA 124,108,108,40,56,16,16,16
2020 DATA 0,7,31,241,31,7,0,0
2030 DATA 16,16,16,56,40,108,108,124
2100 POKE36869,240:PRINTCHR$(147):PRINT"LEIDER VERLOREN"
2110 POKE36878,15
2120 FORI=254TO128STEP-1:POKE36877,I:NEXTI
2130 POKE36878,0:END
2200 POKE36869,240:PRINTCHR$(147):PRINT:PRINT"BRAVO!!!!"
2210 PRINT"SIE HABEN ES GESCHAFFT"
2220 POKE36878,15
2230 FORI=128TO254:POKE36877,I:FORJ=1TO50:NEXTJ,I
2240 POKE36878,0:END

READY.
```